QUERY PROCESSING FROM MILITARY MAPS AND SAND MODELS (GRAPHICAL QUERIES)

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

By

CAPT. D. S. VIRK

to the

COMPUTER SCIENCE PROGRAMME

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST, 1983

Dedicated

To

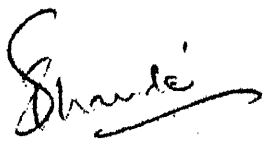
My Parents

CSP-1983-M-VIR-GUE

CERTIFICATE

Certified that the thesis entitled 'QUERY PROCESSIN FROM MILITARY MAPS AND SAND MODELS (GRAPHICAL QUERIES)' by Capt D.S. Virk has been carried out under our supervision and this has not been submitted elsewhere for a degree.

Aug., 1983



(S.G. Dhande)
Asst. Professor,
Department of Mechanical Engineering and
Computer Science Program,
Indian Institute of Technology,
KANPUR

(R. Sankar)
Professor,
Department of Computer Science,
Indian Institute of Technology,
KANPUR

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to Dr. R. Sankar and Dr. S.G. Dhande for their excellent guidance, helpful suggestions, stimulating discussions and encouragement throughout the course of this work.

Thank are due to Capt K.S.N. Choudary, Mr. Deepak Chaturvedi and Mr. A.K. Gupta for their help at various stages of this work.

Thanks are also due to my wife, Minna, for her assistance in preparing the draft and keeping me free from domestic hurdles.

Lastly, I thank Mr. Mohammed Anwar for his excellent typing work.

I.I.T. Kanpur

Capt D.S. Virk

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Computer-aided maps have become a necessity in present day Military applications. Such a system, besides creating, storing and displaying the map, should have efficient query processing capabilities. In Military applications, answers to the graphical queries may be approximated with errors which are permissible within the specified limits. Curves, describing features on the map, are defined by sets of discrete digitized points. These curves can be approximated with different resolutions using Strip Tree concept. Based on this concept, a package has been developed. The package performs graphical query processing pertaining to the features on the map, which are represented by curves, using hierarchical Strip Tree representation. A syntax-oriented command language has been designed for this purpose. Using an existing package for creating the base map, the package developed answers the queries, performing geometrical computations on Strip Trees.

CHAPTER I

INTRODUCTION

1.1 General

With advancement of weapon technology, faster mobility and latest means of communications, modern warfare has become highly complex and technical in nature. Guided missiles, armoured personnel carrier vehicles, computerised communication systems using satellite are some of the examples. It is very vital for a commander to have every possible information about strength and deployment of his forces, their fighting capability and morale, type of weapon systems, guns and fire power available, means of communication with his senior and subordinate commanders, administrative and replenishment facilities available, nature of ground and its topography where he is going to fight his battles etc. etc.

1.1.1 Sources of Military Information

As discussed in preceding paragraph, the nature of information required for military planning is very large and diverse in nature. It is therefore not possible to obtain it from one single source.

In an operational headquarters, such information is collected by different departments and stored in the form of reports, tables, charts, maps etc. For example, the department dealing with military intelligence collects the information from various sources and hands it over to the operation branch. Then this information may be used to update the tables or update the deployment of forces shown on the map or sand models. Subsequently when a commander has to plan an operation the relevant information is extracted from these different sources as and when required.

1.2 Role of Maps and Sand Models

Knowledge of ground, its topography and characteristics is one of the important feature for any military operation. The tactics for battle will depend greatly on this information. Some of the examples highlighting this aspect are:

(i) Hill tops and high grounds are very suitable locations for deploying forces in defensive role.

(ii) An undulating ground like ravines will not be suitable for an operation involving tanks.

(iii) Information regarding roads and tracks is vital to plan movement of troops and maintaining a

supply line.

(iv) Rivers, Nalas and Canals are obstacles in the advance. The information about bridges, the fordability, current flow etc. are important for planning of crossing these obstacles by different means.

Based upon this information, as well as other information of tactical nature, the commanders decide the deployment of their forces in detail. The positions of operational headquarters, which maintain the command and control of battle, various bodies of troops, guns, tanks etc. are decided. However, all these deployments are very dynamic in nature and undergo changes very frequently. The two commonly available methods to project the updated locations on the ground are:

(i) Maps and Sketches

(ii) Sand Models.

1.2.1 Maps and Sketches

Maps are generally the standard documents published by Survey of India. These are available in different scales. The details of information available and accuracy will depend upon the scale of the map. Such base maps provide information of permanent nature e.g. Roads, Tracks, Hills and Highgrounds by Contour Lines, Rivers, Canals,

Villages and Towns, Forests with type of vegetations etc. etc. The military information is then super-imposed upon these maps using different symbols. Similarly, the sketches are made, of the area of interest. However, this may show only the military information with bare essential ground information.

1.2.1.1 Advantages and Disadvantages of Maps

The advantages of using maps are:

(i) Maps provide fairly accurate and detailed information of ground.

(ii) It provides the commander, a 2-dimensional pictorial view in the form of symbols, of the area of operation.

(iii) It is convenient for storage.

The disadvantages of the maps are:

(i) The updating is not very convenient. As discussed earlier, due to rapidly changing situations the deployment has to be changed on these maps.

(ii) The maps are 'cluttered' with unwanted information. The required information has to be extracted from it every time it is required.

1.2.2 Sand Models

These are 3-dimensional extension of maps on a limited scale. A sand model is made using sand, clay, wooden and plastic models of various features and military symbols. Sand model provides a 3-dimensional view of the battle ground. The tactical military plans can be discussed by commanders on these models.

1.2.2.1 Advantages and Disadvantages of Sand Models

The sand models enjoy certain advantages over the maps. These are:

(i) It provides a 3-dimensional view of the ground, although on a limited scale.

(ii) Only such information, which is relevant to the discussions, is depicted; thus highlighting the features of interest and deleting those which are irrelevant.

The drawbacks of a sand model are:

(i) The biggest drawback of a sand model is its bulk and size. It requires a large quantity of stores and a large space on the ground to construct it. However this problem is not faced where the sand models are not required to be shifted, e.g. in training institutions and static headquarters.

(ii) Its construction is time consuming and laborious.

(iii) Because of point at Ser. (ii) above, the modification when area of operation changes, is also very tedious.

(iv) Accuracy is limited, and certain features can not be depicted to the proper scale.

1.2.3 Limitation of Maps and Sand Models in Providing Non-graphical Information

So far we have discussed only those features of information on maps and sand models which can be depicted on them. There is also certain type of information associated with entities on the ground, which can not be depicted on the map or sand models, e.g. the load carrying capacity of a bridge, population of a village or town, unsuitable ground for tank movement etc. etc. We can thus partition the entire information system into two parts:

(i) Graphical information

(ii) Non-graphical information.

1.3 Simulation of Sand Model on Computer

We have seen that present method of using conventional maps and sand models is not comprehensive and fast enough. Whereas the volume of information has increased,

the reaction time available with commanders has decreased due to fast moving battles of today. Use of computers to simulate maps and sand models seems to be the most appropriate solution to this situation.

Computer-aided cartography has been developed in recent years to produce maps. With latest developments in hardware and software, soft copy images on CRT have replaced printed maps for many applications [1]. Automation is also changing the quality as well as amount of information that can be provided. A computer aided system gives advantages of both, the map and the sand model.

Some of the features that a computer based sand model can provide are:

(i) Digitising a given map and storing it on a storage device of a computer.

(ii) It can display the map in any selective manner [2]. It is possible to display a layer of the map which depicts only the roads or only military symbols.

(iii) Any desired area can be enlarged to provide detailed information.

(iv) Updating, corrections and editing is much simpler and faster.

(v) The system will provide a central source of all the data, which can be selected as per requirement.

(vi) It can cater for both graphical and non-graphical data.

(vii) Query answering will be faster and more reliable.

(viii) Can provide necessary security where only authorised persons can have access to data.

(ix) Using animation with raster graphics, movement of troops, contingency and different plans can be displayed [3].

(x) Using 'soft copy' display on CRT devices, it is not necessary to print the map everytime, which saves time.

1.4 Information Retrieval and Query Answering

The main purpose of computer-aided sand model is not only to create and depict a map, but to retrieve any relevant information and answer queries. The effectiveness of computer-aided maps will depend upon the capability to extract information from it, in most efficient manner.

1.4.1 Graphical Data and Queries

All such data which is useful in the graphical display of map on one of the display devices of computer, comes in this category. This data will generally be in the form of digitised data of entities like roads, rivers, contours, and symbols like bridge, temple, survey tree etc. This data deal with only 2-dimensional graphical queries involving points, straight lines and curves as entities.

Similarly the queries of graphical nature are those which make use of such data and perhaps make some computations from this data to answer the query. Examples of this nature are:

(i) Distance of a road from a given point.

(ii) Intervisibility of two points, making computation of heights from contour lines.

(iii) Width of a river at given crossing point.

1.4.2 Non-graphical Data and Queries

All such data which does not pertain to the display in any form comes in this category. This is designed in the form of a data base [4]. This data base is used to answer non-graphical queries, some of which are:

(i) Whether a given road is under military control or civil control.

(ii) Frequency of trains on a given railway line.

(iii) Whether the water source is fit for human consumption or not.

A query language and data base has been designed for this purpose [4].

1.5 Purpose of This Study

Certain amount of work has already been done towards generation of computer-aided map and sand model generation. Maj L.K. Chopra, developed a package which can show interactively, the deployment of military forces [3]. Capt A.V. Subramanian's package generates a base map from digitised data [2]. It also has the facility to create, post and delete the symbols, and displays the map at different layers.

This study deals with graphical queries pertaining to those entities which are represented by curves. It is imperative that the system should be fast in making any computations, at the same time using minimum of storage space. If entire data is stored in the main memory of the computer, no doubt it will be very fast, but the amount of data, when a set of maps is used, will be very large to be stored in the main memory. Therefore proper data structure is to be considered.

Certain amount of approximation of curves can be made whenever the accuracy of query permits. This greatly reduces the time complexity of computation. Methods of approximation and a hierarchical representation using Strips and Strip Tree respectively is implemented. The same has been discussed in succeeding chapters. Chapter II discusses various techniques of approximation of curves by breaking them into strips. Chapter III deals with hierarchical representation of strip trees in the form of a binary tree, and various operations which can be carried out on this structure. Chapter IV gives data structure and implementation details while Chapter V illustrates a case study, based upon the map generated [2].

A list of graphical queries which are discussed in this study is given in Appendix A.

CHAPTER II

CURVE APPROXIMATION BY STRIPS

2.1 General

Two types of information in a map as discussed in previous chapter are, graphical information associated with the display of map, and non-graphical one, which does not pertain to any display but is required to answer queries.

In the remaining chapters we shall discuss those graphical entities on the map, which are represented by curves. These entities are:

(i) Roads
(ii) Cart Tracks
(iii) Rivers
(iv) Canals
(v) Railway Lines
(vi) Boundaries of specified areas e.g. Forest, Marsh Land, Lakes etc.
(vii) International Boundary and Boundary demarking own and enemy forces.
(viii) Contours.

2.2 Requirement of Approximation of Curves

A base map is generated by computer by digitising these entities represented by curves. (In rest of the chapter a curve will mean one of the entities represented by curves.) A digitiser picks up discrete points on a curve and stores them as an ordered list of discrete points. The number of discrete points picked up will depend upon the resolution of digitiser, behaviour of curve and accuracy required. Accuracy of representation of curve increases with the number of points.

In order to answer any query pertaining to a curve, computations may be necessary. Since curves in a map are random in nature and not well behaved regular curves, it is difficult to represent them by well defined mathematical relations. In order to make any computation, it has to be done on the set of points representing it. Considering a digitiser with good resolution, and a curve of fairly long length, the number of discrete points to represent it, may be quite large. Under such circumstances, when computations are made on such a curve, considering set of all the points, it may consume a large amount of time. For example, if we have to find the length of a curve between two given points, it may involve calculation of distance between two successive points and adding them up till the other end is reached. Here time complexity of

computation is of the order of O(n) where n is the number of discrete digitised points between the two given points of the curve.

In military applications of maps, it is not always necessary to make these computations very accurately, and certain amount of error due to approximation may be acceptable. This level of error will depend upon the purpose for which this computations are made. For example, distance between two points, as required for different purposes can be as follows:

(i) In order to site a gun position, and to direct artillary shelling on a known enemy position, accuracy required may be within a few yards.

(ii) In order to plan a VHF radio link between these two points, an error of few kilometers may be acceptable.

However, this margin of error should be left at the discretion of the person using it.

The strip curves provide a method to approximate the curves at different resolutions.

2.3 Strip Curves

In order to save computational time, the effort is to reduce the number of discrete points on the curve,

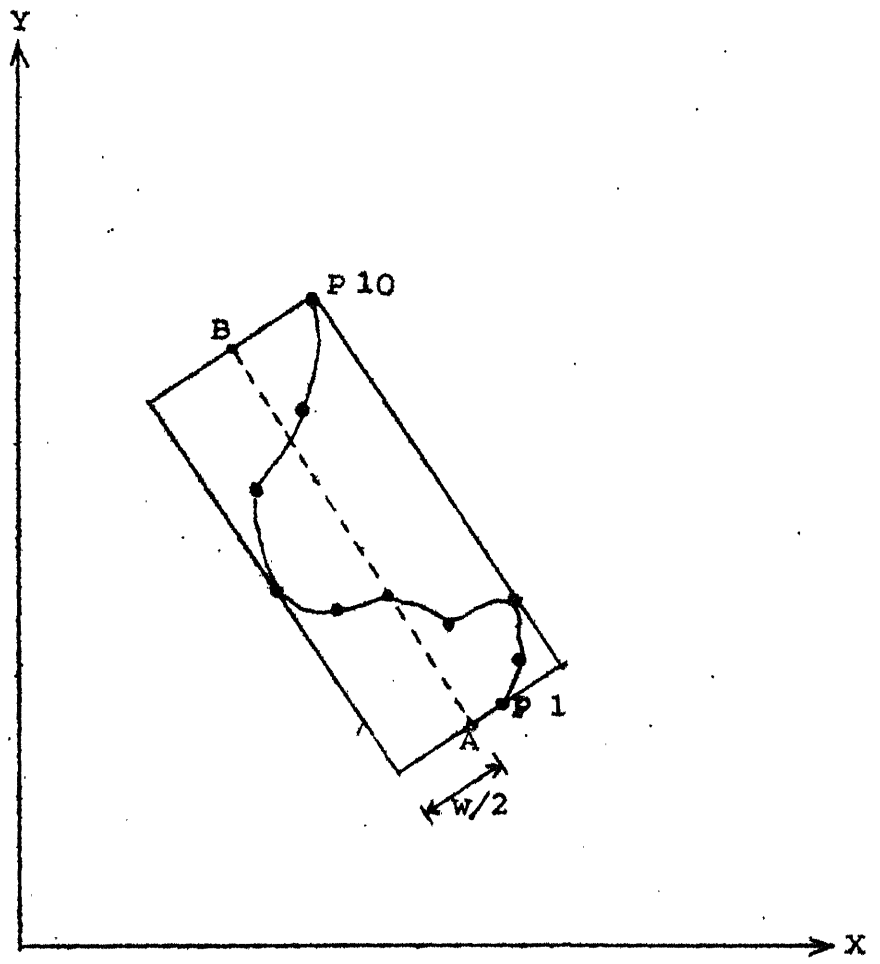


Fig 2.1: Approximation of a Curve by a Strip

sacrificing the accuracy to an allowable limit. Concept of a strip is to approximate a curve by a straight 'line' of thickness w, where w/2 is the maximum allowable error. Fig. 2.1 shows such a representation. A strip is a rectangle enclosing all the points representing the curve. The strip with minimum width is called an Optimum Strip, which gives minimum error at that level. If the accuracy required w_r is less than w/2 provided at this level, the width of strip can be reduced by spliting the set of points (P_1, P_2, ..., P_n) into 2 or more disjoints subsets such that each subset of points results into an optimum strip of width w_r or less. The methods of such sub-division of set of points is discussed in Chapter III.

2.3.1 Advantages of Strip Curves

With this approximation, the points enclosed in the strip are virtually reduced to only two. In Fig. 2.1, points (P_1, P_2, ..., P_{10}) are reduced to points A and B and, the curve is equivalent to only two discrete digitised points A and B. Now if the error of w/2 is allowed for certain application, computation from these two points A & B will be sufficient, saving in time and storage. (The latter we will see in next chapter).

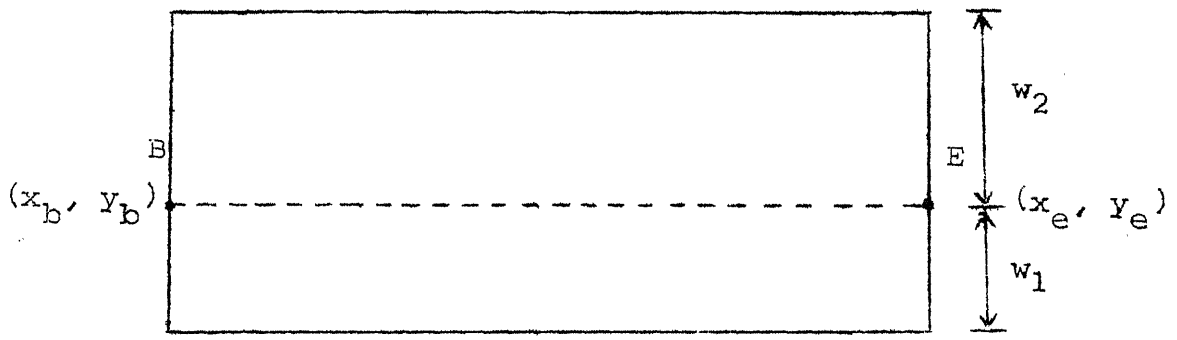


Fig 2.2: 6-Tuple Representation of a Strip Tree

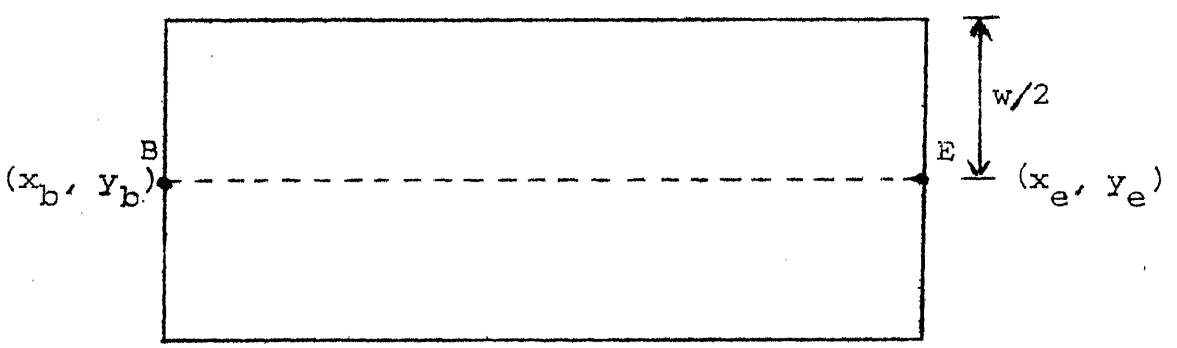


Fig 2.3: 5-Tuple Representation of Strip Tree

2.3.2 Notation of a Strip Curve

In general a strip S can be defined as a 6-tuple [5],

$(x_b, y_b, x_e, y_e, w_1, w_2)$

Fig. 2.2 shows the general form of a strip. The meaning of various terms is as under:

(x_b, y_b) - beginning of line B-E, which is parallel to the longer sides of the strip

(x_e, y_e) - end point of line B-E

w_1 - distance from the B-E to one parallel side of strip

w_2 - distance of line B-E to other parallel side of strip.

(Notation B and E represent points (x_b, y_b) and (x_e, y_e) respectively).

Using this notation, a curve which is enlosed by this strip can be approximated to the line B-E of thickness w_r. In this case the maximum error for any point lying on the curve will be equal to greater of w_1 or w_2. A curve can be represented at resolution w_r if there exists an ordered sequence of m strips

$S_k \qquad k = 0, \ldots\ldots, m - 1$

Such that

$$(w_1 + w_2)_k < w_r \qquad k = 0, \ldots\ldots, m - 1$$

$$x_i \in \bigcup_{k=0}^{m-1} S_k \qquad i = 1, \ldots\ldots, n$$

Where S_k is defined by an appropriate algorithm to divide the set of n points,

$$x_i \qquad i = 1, \ldots\ldots, n$$

into disjoint subsets. These methods are discussed in next chapter.

In above definition, no claim is made of optimization of strip to give minumum of error at a given resolution. The value of error will depend upon the values of w_1 and w_2, which in turn will depend upon the method used to form the strip. These methods are discussed later in this chapter.

Another representation of the strip can be made by a 5-tuple representation -

(B, E, w)

as shown in Fig. 2.3. In this notation, the line used to approximate the curve has been so taken that it is equidistant from the two longer parallel sides of the rectangle forming the strip. In this case the maximum error will only be w/2. Using this notation, the

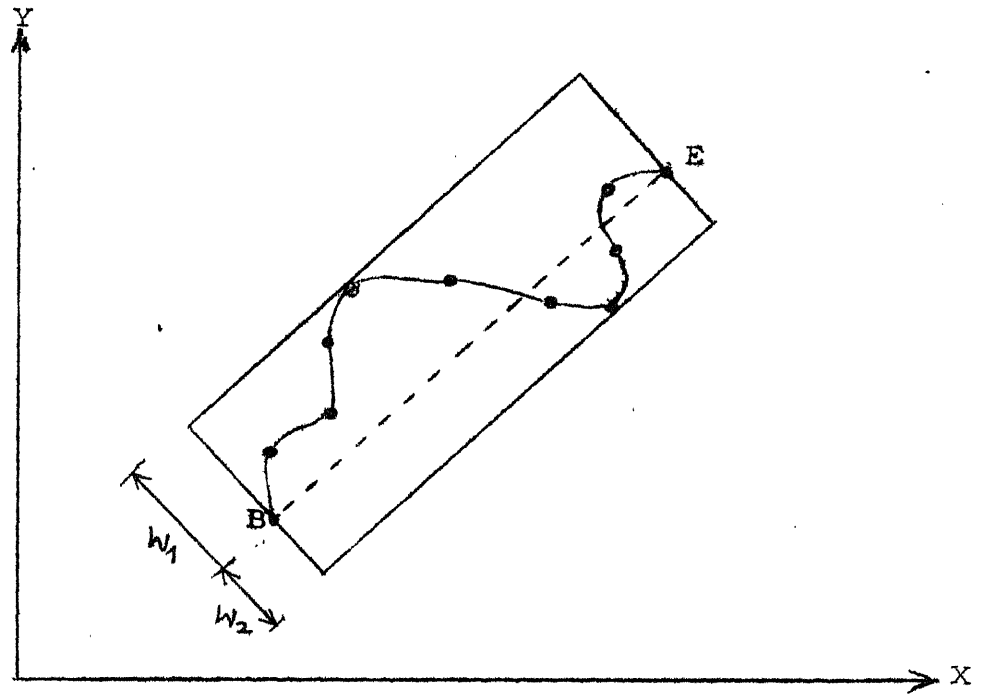


Fig 2.4: Ballard's Method of Creating a Strip

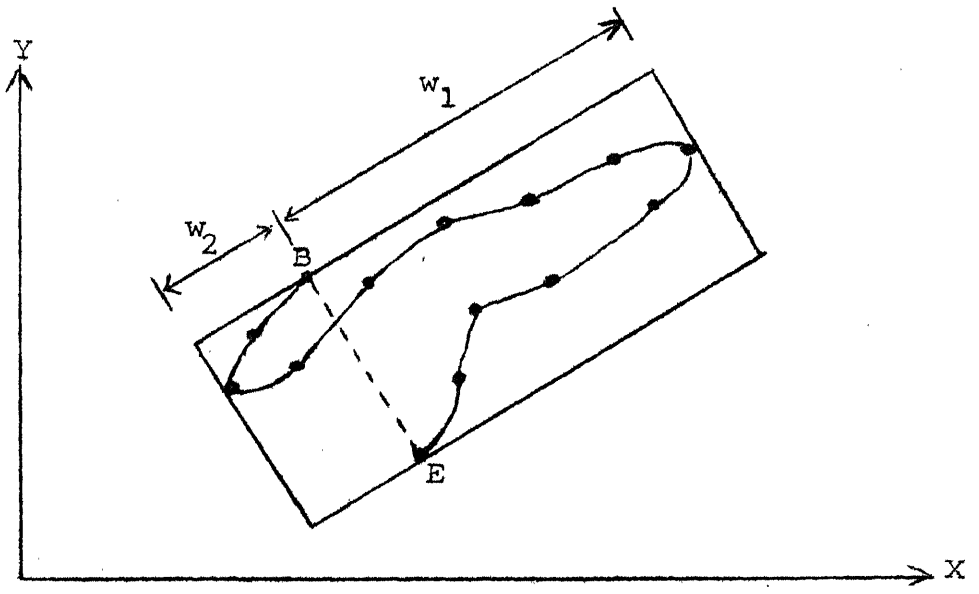


Fig 2.5: Drawback in Ballard's Method

computations are more convenient, efficient and it enables the Strip Tree structure with better efficiency, as we will see in next chapter.

2.4 Methods of Constructing a Strip

Dana H. Ballard [5] has discussed Strip Tree structure giving one algorithm for construction of strips. Three methods discussed in this study are:

(i) Ballard's method

(ii) Least Square distance method

(iii) Optimum strip method

2.4.1 Ballard's Method

The line approximating the curve is obtained by joining first and the last points on the curve. (Fig. 2.4). This gives the general direction of the curve. The strip is formed by finding smallest rectangle with two sides parallel to this line, and just covering all the points on the curve. This method uses six-tuple notation of strip and maximum error will be equal to the distance of the line from the parallel side of the strip which is farthest.

This method, although simple in its implementation, gives quite undesirable results in certain odd cases. An example of this is shown in Fig. 2.5. Here line B-E,

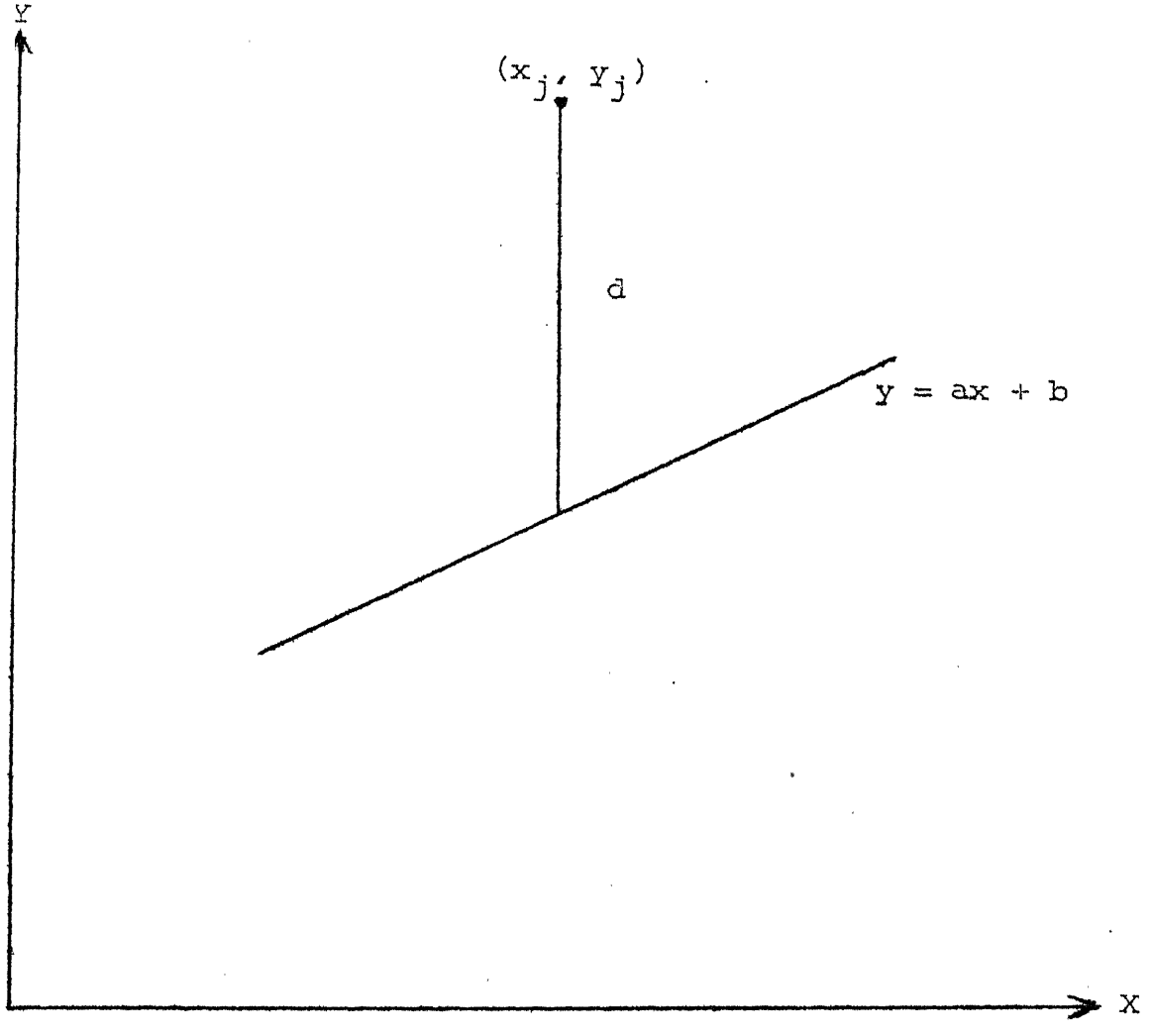


Fig 2.6: Least Square Method of Creating Strip

approximating the curve, does not give a very good approximation, as also the error will be w_1, which is very large compared to length of the curve.

2.4.2 Method of Least Squares

Erwin Kreyszig has discussed the method of least square Distance [6] to fit a line through a set of data points.

Consider a set of n points

$$(x_1, y_1), (x_2, y_2), \ldots\ldots \quad (x_n, y_n)$$

assume that line

$$y = a + bx$$

is the line passing through the set of points. Vertical distance of any point (x_i, y_i) from this line is given by (Fig. 2.6)

$$d = y_j - a - bx_j$$

Then the sum of square of all the distances is given by

$$q = \sum_{j=1}^{n} (y_j - a - bx_j)^2$$

The value of q is dependent on values of a and b. For value of q to be minimum,

$$\frac{\partial q}{\partial a} = 0 \qquad \text{and}$$

$$\frac{\partial q}{\partial b} = 0$$

which gives us the equation of 'Regression Line' as

$$y - \bar{y} = b(x - \bar{x}).$$

This line passes through point $(\bar{x}, \bar{y})$ with slope b, which is called the 'Regression Coefficient' of y on x, and values of $\bar{x}$, $\bar{y}$ and b are given by relations

$$\bar{x} = \frac{1}{n} (x_1 + \ldots\ldots\ldots\ldots x_n)$$

$$\bar{y} = \frac{1}{n} (y_1 + \ldots\ldots\ldots\ldots y_n)$$

$$b = \frac{Sxy}{S_1^2}$$

Sxy is called 'Convariance of Sample'. Here

$$S_1^2 = \frac{1}{n-1} \sum_{j=1}^{n} (x_j - \bar{x})^2$$

and $$Sxy = \frac{1}{n-1} \sum_{j=1}^{n} (x_j - \bar{x})(y_j - \bar{x})$$

Therefore by computing values of S_1, Sxy we can compute slope of the line which passes through $(\bar{x}, \bar{y})$. The strip can then be formed as in Ballard's Method by

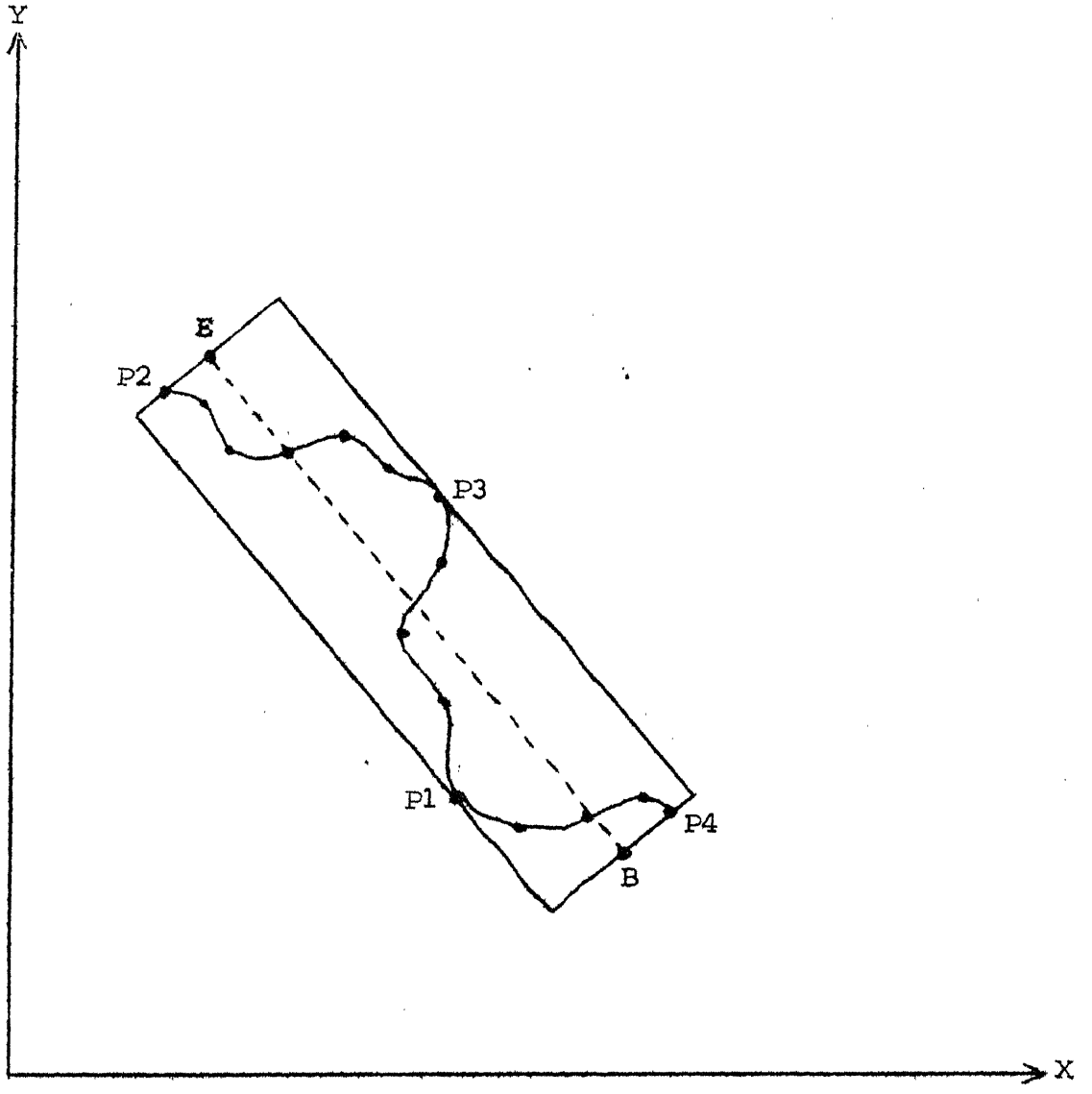


Fig 2.7: Optimum Strip Method

finding two sides parallel to this line and just enclosing all the points. This also uses the 6-tuple notation of the strip. It was experimently seen that strips such formed are not optimum strips.

2.4.3 Optimum Strip Method

An optimum strip has been defined as that strip which has a minimum width for given set of points defining the curve. By ensuring that the width is minimum, the margin of error is reduced to minimum. The method uses the notation of 5-tuple representation for the curve. Fig. 2.7 shows an optimum strip constructed by this method.

Like previous method of Least Square Distance, the shape of curve has no meaning, only the set of points describing the curve are considered.

This method consists of two parts:

(i) To determine the direction of optimum rectangle

(ii) To construct the strip.

2.4.3.1 To Find Direction of Optimum Strip

This involves computing smallest width for the given set of points, at different angles and comparing the results.

Consider a set of points X_1, X_2, ..., X_n describing a curve, and a line

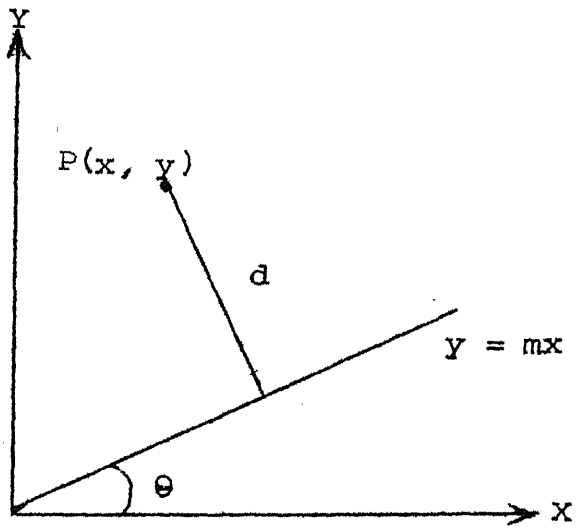


Fig 2.8: Finding Minimum Width for a given Direction

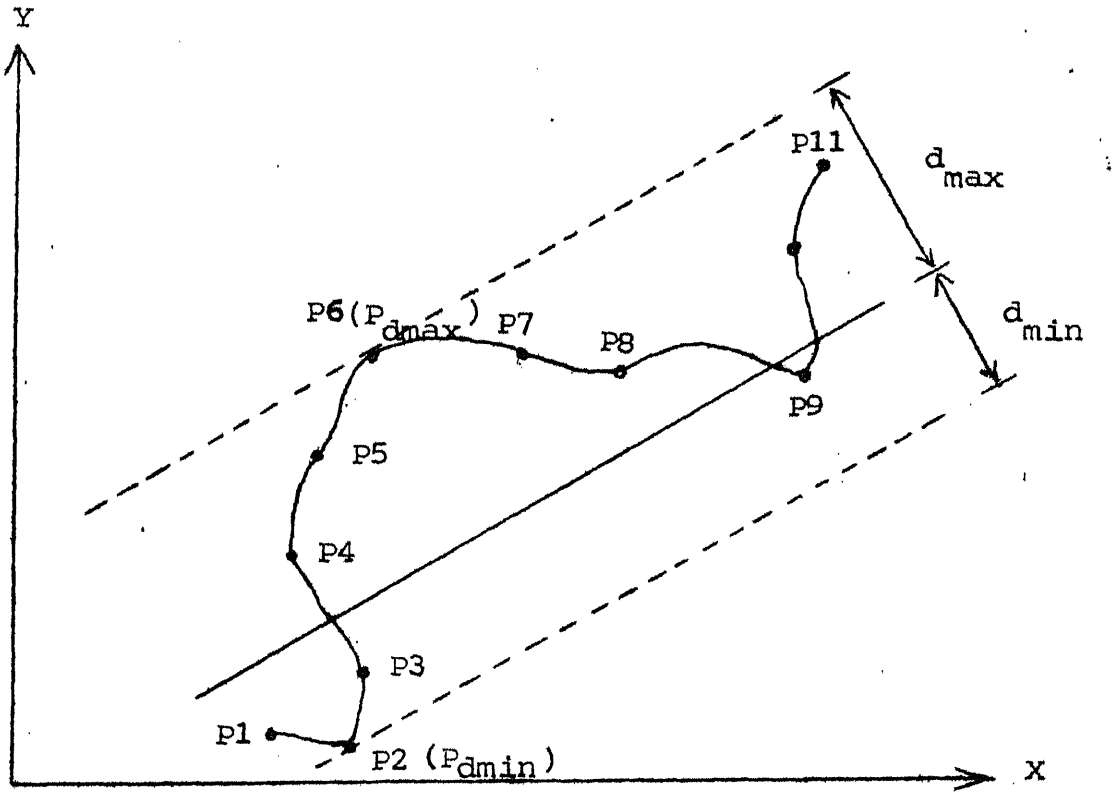


Fig 2.9: To Find Farthest Points on Two Ends from a given Line. P2 and P6 Lie at the Two Farthest Limits

$$ax + by + c = 0.$$

If we compute perpendicular distance of each point from this line, then the two points which are maximum and minimum distance from this line respectively, mark the two boundaries of the strip. The difference of these distances d_{max} and d_{min} give the width of strip which is parallel to this line, the two sides passing through P_{dmax} and P_{dmin} respectively. Fig. 2.8 illustrates this method. In Fig. 2.9, point P_6 is at max distance from given line, and point P_2 at min distance. (Only magnitude is not sufficient for this, thus P_8, and P_{10} which have lesser magnitude of distance but with +ve sign are at greater distance than that of P_1). Therefore, the two dotted lines mark the two parallel sides of the strip which is constructed in the direction of given line. The next step in this part involves repeting this iteration for lines at different inclinations. The algorithms for chosing steps and different values of directions is discussed in next chapter.

For the ease of computations, we consider the lines which pass through the origin of the coordinate system. Thus we have lines

$$y = mx$$

for different values of m. Distance of a point (x_1, y_1) from the line

$$ax + bx + c = 0$$

is given by

$$d = \frac{ax_1 + by_1 + c}{\sqrt{a^2 + b^2}}$$

but if the line is $y = mx$, passing through the origin, then

$$a = m = \text{Tan } \theta$$

$$b = -1$$

(refer to Fig. 2.7). Then the distance is given by

$$d = x_1 \sin \theta - y_1 \cos \theta$$

By changing the value of θ from 0° to 180°, in the steps as determined by suitable algorithm, we can compare the width of strip in each direction. The strip having minimum width will give us the direction of optimum strip, $\theta_{optimum}$.

2.4.3.2 Constructing Optimum Strip in $\theta_{optimum}$ Direction

Having found the direction of Optimum strip, we get two points through which pass the two sides of rectangle which are parallel to the direction. The other two sides of the rectangle can similarly be found by finding the two points which are at maximum and minimum distance respectively from a line which is perpendicular to the

direction of strip. Line B-E approximating the curve is obtained by joining the midpoints of these two lines which are obtained in the direction perpendicular to the optimum strip direction. Fig. 2.7 shows an optimum strip with four boundary points, P_1 and P_3 lying on the sides which are parallel to the direction, P_2 and P_4 on the sides perpendicular. Line B-E will approximate any curve which has points lying within this rectangle and includes P_1, P_2, P_3 and P_4. Therefore we observe that ordering of points, or in other words shape of curve has no effect on the strip.

2.4.4 Comparison of the Three Methods

The basic purpose of using strips is to approximate a curve such that it reduces the computation and storage effort, at the same time with minimum possible error. In Ballard Method, the error in worst cases, may be very large as illustrated in Fig. 2.5. This does not give a true approximation of the curve.

The Least Square Distance method is improvement in Ballard Method where shape of curve is neglected but only the set of points on the curve are considered. However, it involves larger amount of computations, and may still not give an optimum strip.

The optimum Strip Method also involves larger amount of computations, computing distance, of all the

points from lines at different inclination, but it gives the best approximation to the curve. Although there might still be a quantisation error due to steps of angle from 0° to 180° chosen for various lines, but this error will be negligible if these steps are chosen properly.

In this study the method of Optimum Strip has been used to construct the strips. It's implementation details are discussed in succeeding chapters.

CHAPTER III

STRIP TREES AND BASIC OPERATIONS ON STRIP TREES

3.1 General

In Chapter II, methods of approximating a curve by a strip were discussed. The strip enclosing the entire curve is called the 'Root Strip'. The width of the Root Strip will depend upon the set of points defining the curve. In case of Ballard's method of forming a strip it also depends upon the position of first and the last point in the ordered set of points. This Root Strip is not of much practical use since the width of this strip is not predicatable, and hence the level of error is also not predicatable. In order to reduce the width of the strip to an acceptable error level, we have to split the set of points to smaller disjoint subsets.

If

$$S = \{p_1, p_2, \ldots\ldots\ldots\ldots p_n\}$$

be the set of points defining a curve, then we can partition these points into smaller subsets

$$S = \{s_1, s_2, \ldots\ldots\ldots\ldots s_q\}$$

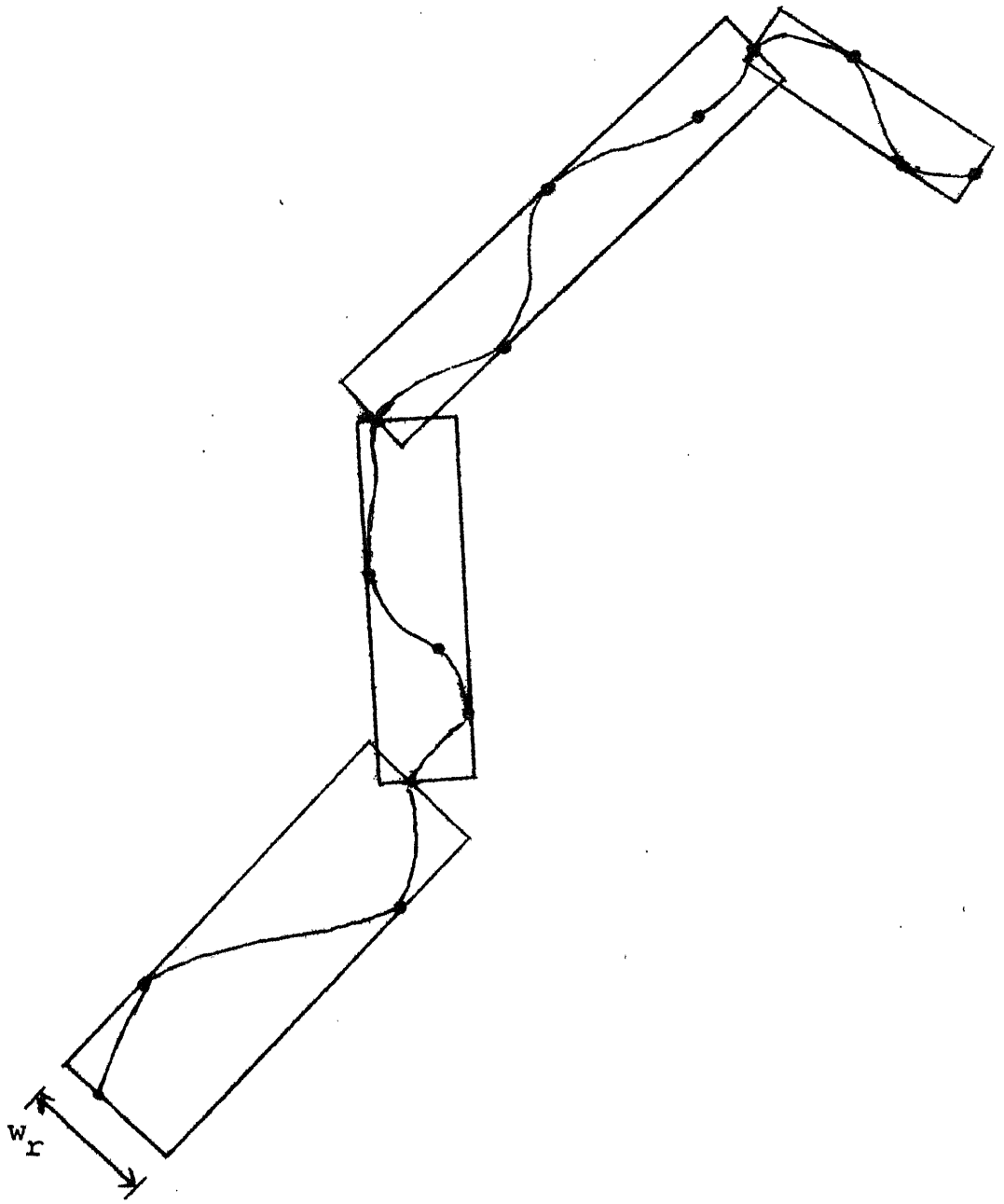


Fig 3.1: Representation of Curve by Strips

such that

$$S_1 = \{ P_1, P_2, \ldots\ldots, P_i \}$$

$$S_2 = \{ P_i, P_{i+1}, \ldots\ldots, P_j \}$$

$$S_3 = \{ P_j, \ldots\ldots\ldots\ldots \}$$

$$\ldots\ldots\ldots$$

$$\ldots\ldots\ldots$$

$$S_q = \{ \ldots\ldots\ldots\ldots, P_n \}$$

and

$$W_{sk} \leq W_r \qquad k = 1\ldots\ldots\ldots\ldots q$$

where

W_{sk} is width of Root Strip for Subset S_k

W_r minimum required width for allowable error in approximation.

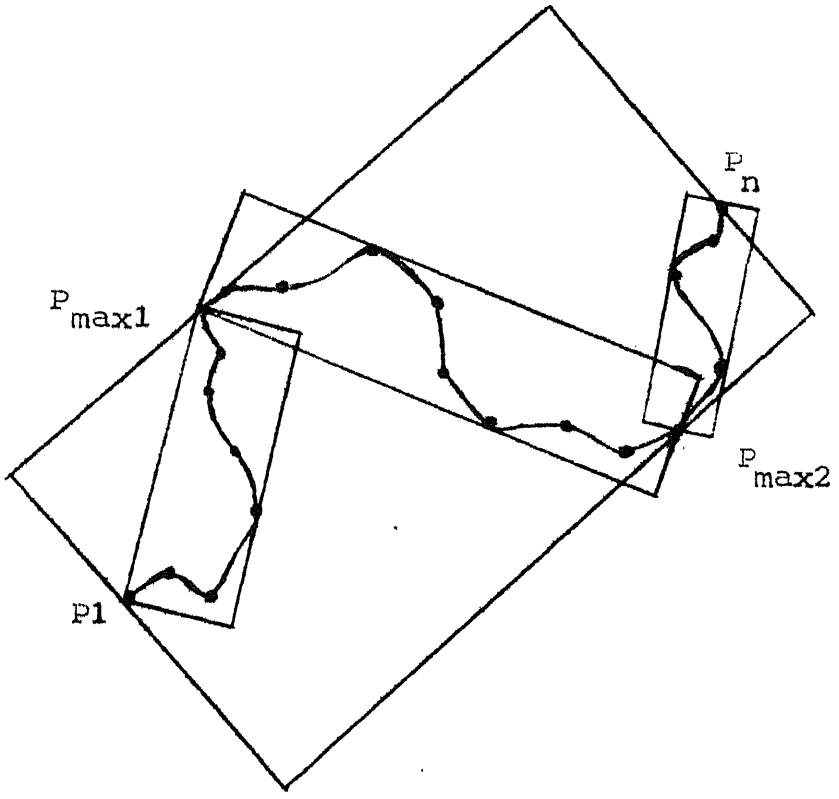
Figure 3.1 shows an example where a curve defined by 15 points has been subdivided into 4 strips, each strip having a width less than or equal to W_r the required minimum width. The curve, which was represented by 15 points can now be represented by these four strips. If still lower resolution, say $W_r/2$ is required, then first, second and third strips will be again subdivided into strips of lesser width.

3.2 Strip Trees

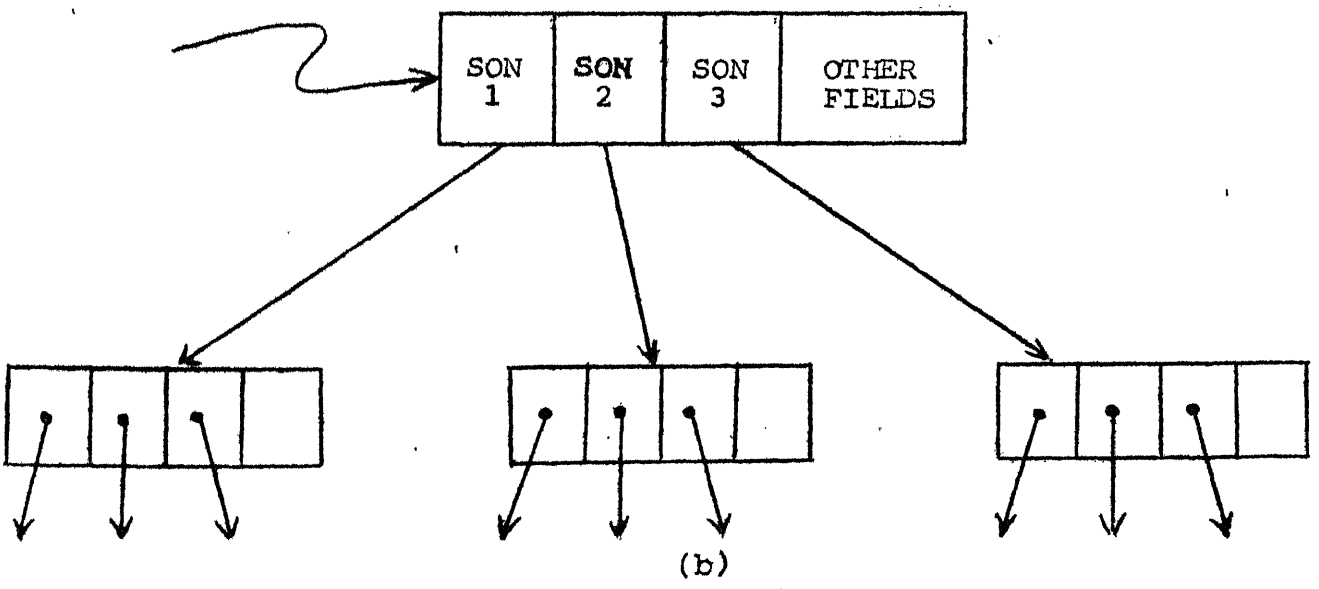
A Strip Tree is a hierarchical representation of a curve in the form of strips, such that each lower level corresponds to finer resolution representation of the curve. This is achieved by considering each subset of points as a new set, and subdividing each set of points whose Root Strip (also called Root Node) has width greater than w_r. This tree structure has properties desirable for various computations required to be carried out on these curves. Intersection and point membership (for closed curves) can be resolved with time complexity $O(\log n)$, where n is the number of points describing the curve. The curves can be efficiently encoded and displayed at various levels of resolution. The representation is closed under intersection and union and these too, can be carried out at different resolutions. All these properties depend on the hierarchical tree structure which allows primitive operations to be performed at lowest possible resolution with great computational time savings.

3.3 Methods of Forming a Strip Tree

Efficiency of Strip Tree structure for computation will depend upon the method of representation of the tree. Theoretically, each strip with width w_r can be



(a)



(b)

Fig 3.2: Tertiary Strip Tree Representation
(a) Sub-division of Strips
(b) Tree Representation

subdivided in n-subtrees, thus forming a n-ary tree structure. Basic consideration in methods of sub-division is that leaf strips of width w_r should be formed as fast as possible, keeping in view the computational complexity in mind. The four methods of sub-dividing a strip are:

(i) Tertiary Strip Tree

(ii) Maxpoint Binary Strip Tree

(iii) Midpoint Binary Strip Tree

(iv) Leaf-Node-First Strip Tree

3.3.1 Tertiary Tree Method

Fig. 3.2 illustrates this method. We have seen in Chapter II, that while forming a strip, each of the four sides of rectangle will pass through atleast one point each on the curve. This is to minimise the size of strip, after the direction of B-E line has been found. Tertiary Strip Tree Method uses two points which lie on the lines parallel to B-E line, to sub-divide the strip.

If strip S is enclosing a curve with points

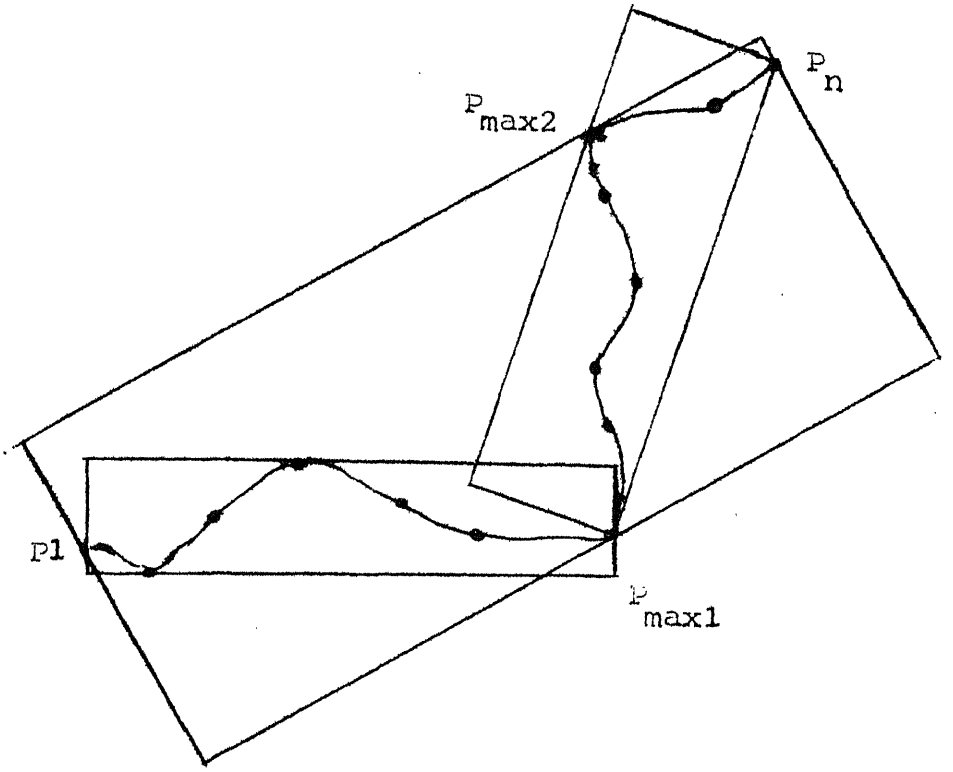
$$S = \{P_1, P_2, \ldots\ldots\ldots P_n\}$$

then its siblnigs will be the strips enclosing points

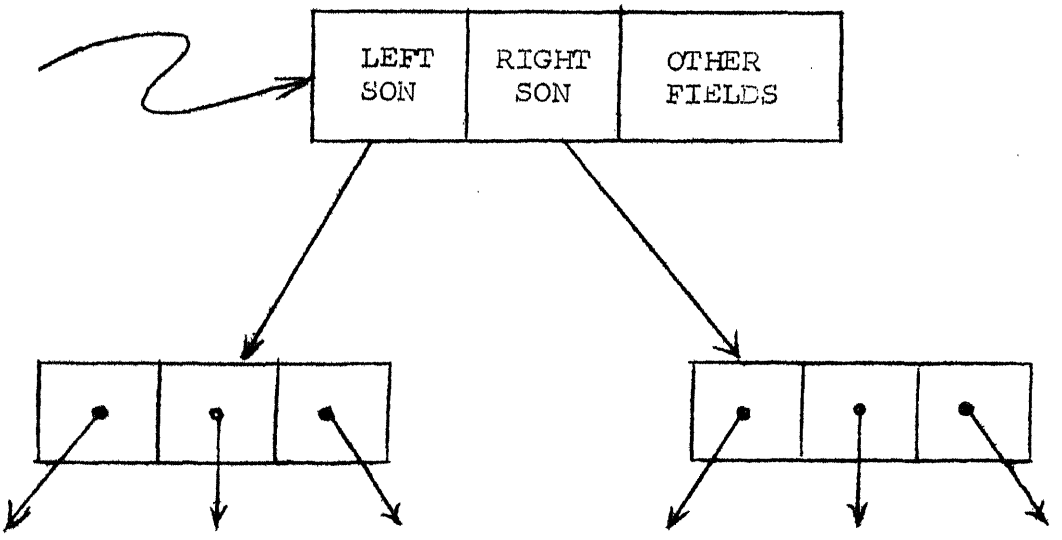
$$s_1 = \{P_1 \ldots\ldots\ldots\ldots P_{max1}\}$$

$$s_2 = \{P_{max1} \ldots\ldots\ldots P_{max2}\}$$

$$s_3 = \{P_{max2} \ldots\ldots\ldots P_n\}$$



(a)



(b)

Fig 3.3: Max-point Binary Strip Tree Representation (a) Sub-division of Strips (b) Tree Representation

where P_{max1} and P_{max2} are the two points on the lines parallel to B-E and P_{max1} is earlier than P_{max2} in ordering of points $P_1 \ldots P_n$.

The method makes use of probability that the curve has a more uniform slope in these three parts than in any other position. Convergence to the leaf strips of width w_r is fairly fast in this method. However, a tertiary tree is more combursome for making computations, although more efficient to form.

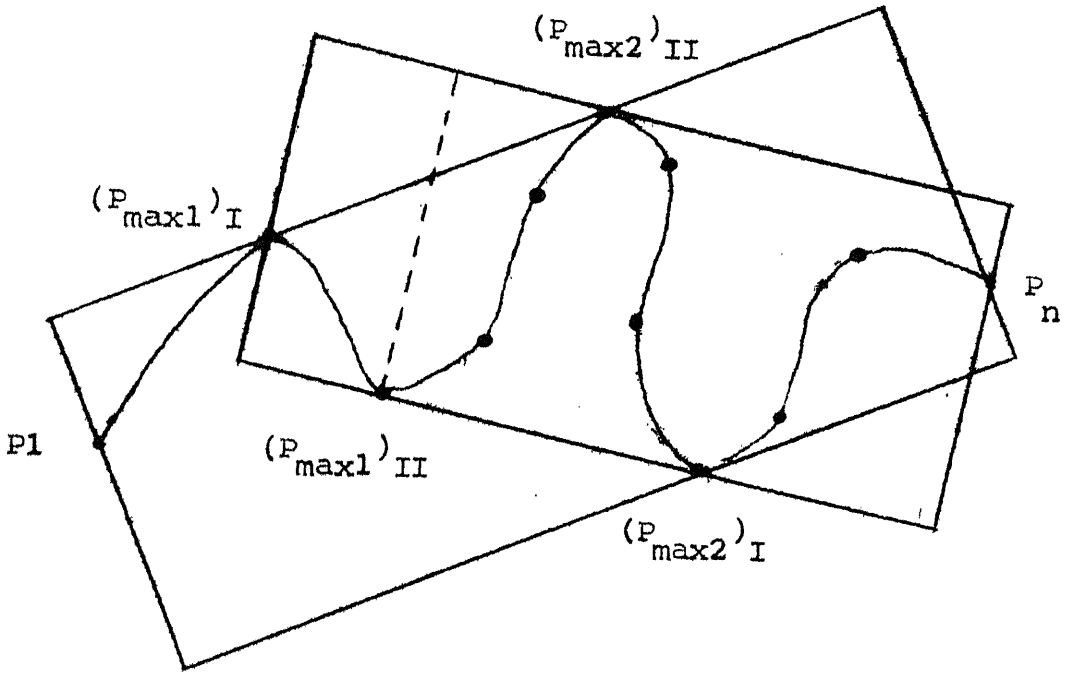
3.3.2 Maxpoint Binary Strip Tree

This method is a Binary-Tree adoption of previous method. In order to reduce the tertiary tree to a binary tree, for ease of computations, it uses one of the two points on the lines parallel to B-E (Fig. 3.3). Here a consideration, as to which of the two points is to be selected as dividing points, has to be made.

In case P_{max1} is such that it comes before P_{max2} in the order of points, and it is selected, then strip for points

$$s_1 = \{ P_1 \ldots\ldots P_{max1} \}$$

will converge faster because it will have lesser width than other one.



$(P_{max1})_I$ & $(P_{max2})_I$ - Limiting points in first sub-division

$(P_{max1})_{II}$ & $(P_{max2})_{II}$ - Limiting points in second sub-division

Fig 3.4: Illustrates Inefficiency of Max-point Strip Tree Method, where P_{max1} is chosen as Dividing Point without Considering its Location.

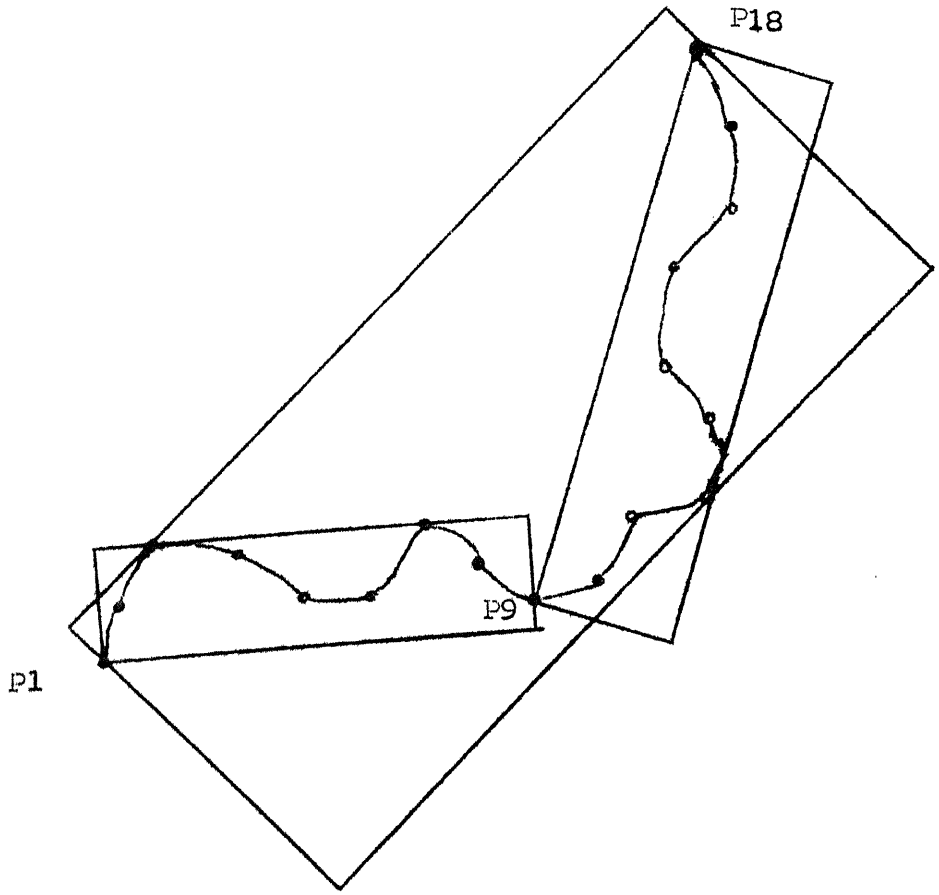


Fig 3.5: Mid-point Sub-division Strip Tree Representation

This method can become very inefficient in the worst case as illustrated in Fig. 3.4. Here in first two iterations, the left subtree has only 2 points, because in each subtree second point happens to be P_{max1}. To avoid this situation, a test can be made on P_{max1} and P_{max2}. The points which lies nearer to the line, perpendicular to B-E and passing through the mid point of B-E should be selected as subdivision point. This will ensure that the point is not close to either first or last point of the curve lying inside the strip. This method has to make extra computations and comparisons thus making it slower.

3.3.3 Mid-point Binary Strip Tree

The method is simplest and therefore fastest in implementation. If the strip has n points then, sub-division point p_d is given by

$$p_d = p_{(\frac{n}{2})} \qquad n \text{ even}$$

$$p_d = p_{(\frac{n+1}{2})} \qquad n \text{ odd.}$$

Therefore both the sub-strips have either equal number of points or a difference of one point. Complexity of this method will be O(log n). (Fig. 3.5)

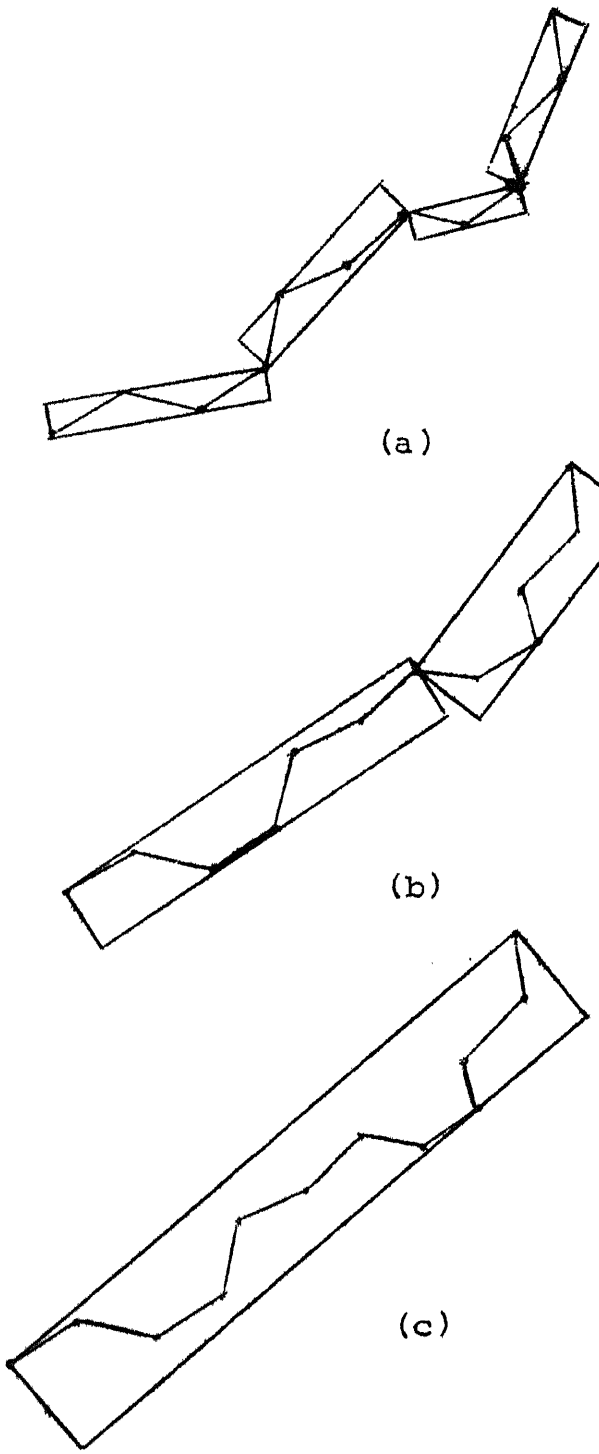


Fig 3.6: Leaf-First Strip Tree Method
(a) Leaf Nodes (b) Intermediate Step
(c) Root Node

3.3.4 Leaf-First Strip Tree

So far we have discussed only top down approach of forming a strip free, where, each strip yields two siblings. This method is reverse of top-down technique, in which two adjoining strips are combined to creat their parent.

Initially, the algorithm starts with constructing leaf nodes, each having width$\leq w_r$. This can be done by selecting subsets

$$s_1 = \{P_1, P_2 \ldots\ldots P_i\}$$

$$s_2 = \{P_i, \ldots\ldots P_j\}$$

$$s_3 = \{P_j, \ldots\ldots\}$$

$$s_p = \{\ldots\ldots\ldots P_n\}$$

such that

$$w(s_k) \leq w_r$$

and $w(s_k$, first point of $s_{k+1}) > w_r$, $k = 1, \ldots\ldots p$, where $w(s_k)$ is width of strip formed for points of subset s_k.

Having formed the leaft nodes, we can form nodes with coarser resolution, by combining points of two succe-ssive nodes and forming a single new node. This itera-tion stops when a single root node is formed. Fig. 3.6 illustrates this, where initially 4 nodes were formed,

and in two successive iterations 2 and single root node is formed.

3.4 Basic Operations on Strip Trees

Computations on a curve can now be performed using the strip Tree formed for the same. The computational complexity of various operations which can be performed, may be difficult to characterize because it depends on the particular geometry of the curve. If the curve is well behaved i.e. if it is relatively smooth, then the algorithms are very efficient. If we are using binary trees, and look at only one of the two sons at any node, then the time complexity is of the order of O(log n). The curves generally encountered in a map fall in this category. There are some closed curves, however, which are not well behaved e.g. contours, boundary of specified area etc. But in such cases, the nature of queries generally are of point containment which, as we will see later, can be computed quite efficiently by using higher level nodes itself.

The operations which can be performed on Strip Trees, and are used in query answering techniques of Chapter IV are given below. This assumes that a binary tree with leaf nodes of width w_r has been created.

3.4.1 Display of Curve at Different Resolutions

As discussed earlier, a curve may be represented as a set of strips such that each strip has a resolution width less than some fixed value w_r. If we have a display device depended procedure STRIPDISPLAY to draw a rectangle, then algorithm to display a curve at resolution w_r will be as given below. (A pseudo-Pascal language is used to describe all the algorithms.)

```
procedure DISPLAY (T, w_r);
  begin
    if w(T) ≤ w_r then STRIPDISPLAY(T)
    else DISPLAY(LSON(T), w_r) and
         DISPLAY(RSON(T), w_r)
  end;
```

3.4.2 The Length of the Curve

The Strip Tree provides a very simple method of calculating the length of a curve at given resolution w_r.

```
function LENGTH (w_r, T):real;
  begin
    if W(T) ≤ w_r then
      LENGTH := (SQRT((s_b(T) - x_e(T))^2 + (y_b(T) -
                                              y_e(T))^2).

    else
```

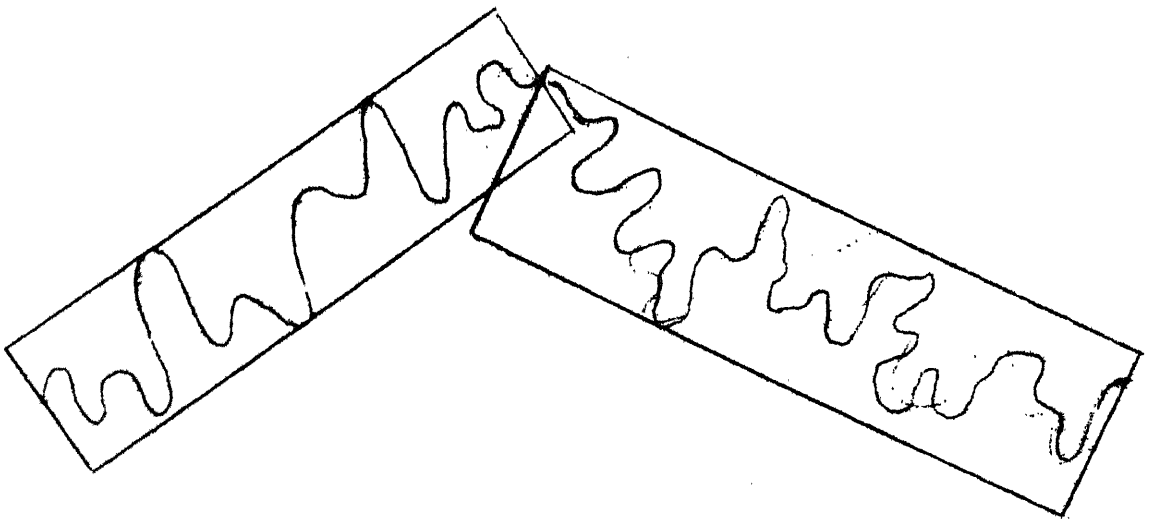


Fig 3.7: Algorithm for Computing Length using Strip Length Causes Unpredictable Error

```
        LENGTH := LENGTH(w_r, LSON(T)) + LENGTH(w_r,
                                          RSON(T)))
    end;
```

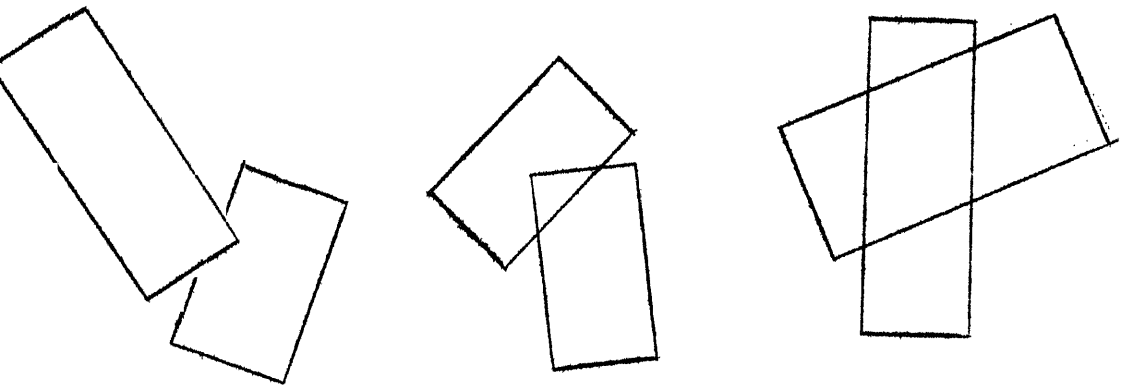
This algorithm sums up the lengths of all those strips which are just equal to or less than w_r, while travelling down the tree. This may cause unpredicable error in case of such curves which have too many short turns, as shown in Fig. 3.7. Once again the assertion is made, that if the curve is well behaved, the approximation may be acceptable. However in actual implementation, a modified algorithm where actual length of curve is stored in each strip, has been used. The same will be discussed in next chapter.

3.4.3 Intersection of Two Curves

One of the important features of the representation is its ability to compute intersection between curves at higher resolution itself. In order to explain the algorithms for intersection, following definitions are necessary.

(a) Two strip segments, S_1 derived from curve C_1 and S_2 derived from C_2 have a '<u>null</u>' intersection iff $S_1 \cap S_2 = 0$.

(b) Two strip segments S_1 and S_2 have a '<u>clear</u>' intersection iff all the sides of the strip



(a) NULL (b) POSSIBLE (c) CLEAR

Fig 3.8: Types of Intersections between Two Strip Segments

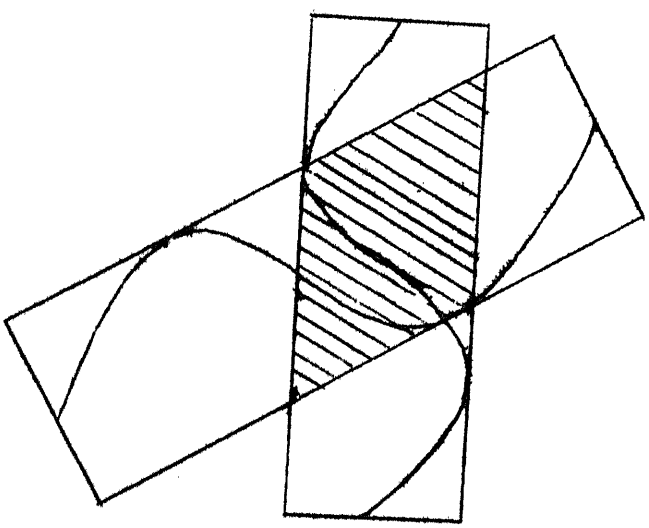


Fig 3.9: If the Two Strip Segments have a Clear Intersection, Corresponding Curve must intersect

parallel to line segment $(B-E)_1$ and $(B-E)_2$ intersect.

(c) Two strip segments S_1 and S_2 have a 'possible' intersection if condition (b) is not satisfied and yet $S_1 \cap S_2 \neq 0$.

These cases have been illustrated in Fig. 3.8. A fairly obvious but very important lemma is given:

CLEAR INTERSECTION LEMMA: If two strip segments have a clear intersection and the strips are both regular, then the corresponding curves must also intersect. Fig. 3.9 shows such an intersection.

From this lemma we can make the assertion that if two root strip segments corresponding to two curves, do not intersect then the curves do not intersect, and if the root strip segments have a clear intersection then the curve also intersects. This important assertion helps in finding intersection at higher levels itself, without having the need to travel the trees to leaf nodes of width w_r or less.

In the algorithm given below, function STRIP-INTERSECTION determines the type of intersection between two strip segments. A(T) is the area of strip at node T.

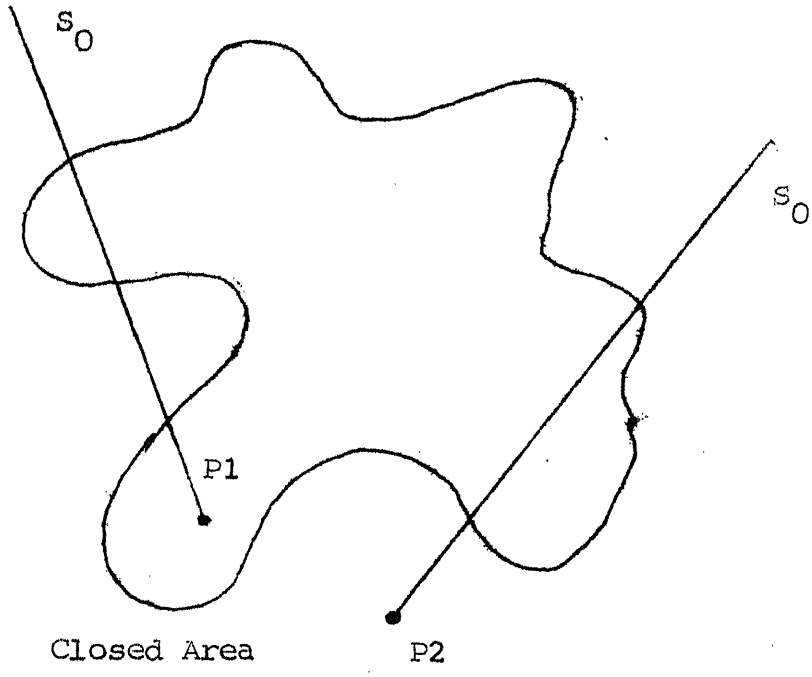


Fig 3.10: To test whether a Point is inside a Closed Area

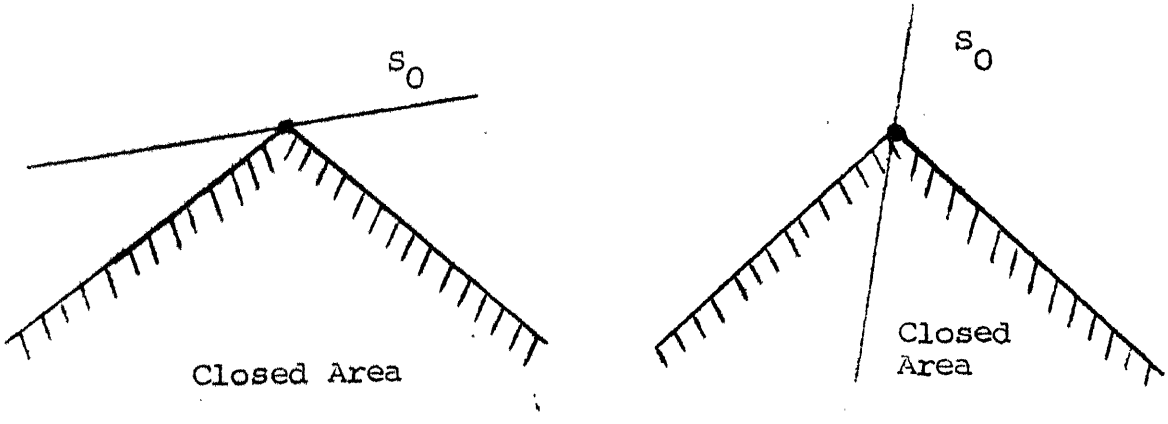


Fig 3.11: Cases to illustrate Ambiguities in Testing Containment of a Point in a Closed Area.

```
function INTERSECTION (T1, T2): boolean;
  begin
    Type := STRIPINTERSECTION(T1, T2);
    Case Type of
    Null: INTERSECTION := false;
    Clear: INTERSECTION := true;
    Possible: if A(T1) > A(T2) then
      INTERSECTION := (INTERSECTION (LSON(T1), T2)
      OR (INTERSECTION(RSON(T1), T2)
      else
      INTERSECTION := (INTERSECTION(T1, LSON(T2))
      OR (INTERSECTION(T1, RSON(T2))
   end; {case}
 end;
```

3.4.4 Determining whether a Point is inside a Closed Curve

The Strip Tree representation of an area by its' boundary allows in a simple and straight forward manner, determination whether a points is inside a closed curve. Following Lemma is used in the algorithm for determining the containment of a point [7].

POINT CONTAINMENT LEMMA: If any semi-infinite line terminating at the point intersects the boundary of an area an odd number of times, the point is inside the area. (Fig. 3.10)

To decide whether a point P is member of an area represented by a Strip Tree, we need only to compute the number of clear intersection of the Strip Tree with any semi-infinite strip, S_0 which has width 0 and emanates from P. If this number is odd then the point is inside the area. The algorithm given below uses a procedure CREATSTRIP (S_0, P) which creats a strip for semi-infinite line emanating from P.

```
function INSIDE (P, T): boolean;
  begin
    CREATSTRIP(S0, P);
    NUMBER := NOINTERSECTIONS (S0, T);
    if NUMBER is odd then INSIDE := True
      else INSIDE := flase
  end;
function NOINTERSECTION (S, T) : integer;
  begin
    TYPE := STRIPINTERSECTION (S, T);
    Case Type of
      NULL: NOINTERSECTION := );
      CLEAR: NOINTERSECTION := 1;
      POSSIBLE: NOINTERSECTION := NOINTERSECTION
      (S, LSON(T)) + NOINTERSECTION(S, RSON(T))
    end; % case \
  end;
```

This algorithm runs into trouble when strip S_0 is tangent to the curve (Fig. 3.11(a)). Similarly if strip S_0 passes through an end point of a node in Strip Tree corresponding the curve, number of clear intersection will be 0, causing an ambiguity. (Fig. 3.11(b)). These ambiguities can be, however, overcome by more complex algorithms.

3.4.5 Other Operations

Besides the operations discussed so far, certain other operations which can be performed on Strip Trees are:

(i) Intersection of a curve with an Area

(ii) Intersection of Two Areas

(iii) The union operations.

(i) and (ii) are essentially versions of intersection of two curves where Areas are described by closed curves. The operation of union has not been used for any of the query answering algorithms of Chapter IV, and therefore it has not been discussed here.

CHAPTER IV

ALGORITHMS, DATA STRUCTURE AND IMPLEMENTATION

4.1 General

In Chapter III, techniques of forming Strip Trees and certain basic operations which can be performed on Strip Trees, were discussed. In this chapter we shall discuss the design of a package and its implementation for processing various graphical queries making use of Strip Trees. The scope of queries is restricted to Military applications only.

A compromise between efficiency in time and storage space is essential for any such system, and a suitable Data Structure is necessary for the same. Dynamic storage allocation, using linked list, has been used wherever possible. To make the package, user friendly, a suitable command language has been designed.

4.2 Scope and Capabilities of Package

The package is developed to process all such graphical queries of Military nature, which can be answered using existing graphical data on a computer-aided

map. This data is essentially in the form of curves describing various features on the map, as discussed in earlier chapters.

The queries which can be processed are:

(i) Distance.

(ii) Length of feature.

(iii) Containment of a point within a closed area.

(iv) Area of closed areas.

(v) Height of a point on the map.

(vi) Intervisibility between two points.

(vii) Magnetic, Grid and True bearings from a point to another point.

(viii) Intersection of features.

The various modes of these queries are possible. A list of such modes and corresponding user commands, is listed at Appendix 'B'.

The package has been designed to be compatible with map generation package earlier developed [2]. It uses the same input source for creating Strip Trees for various features, which has been used to generate the map. This input is in the form of digitised data for curves, available in source file DATAIN. A serial number has been allocated to each feature within the same class of features. This enables package to distinguish from one feature to another.

The command language gives the flexibility to user to give arguments to query in any order (except in certain cases). An error detection system, embedded in the package, can report the error to user at different stages. Package accepts all commands in strings form, to avoid error in data type in input. The error in data type, if any, is detected by the program and appropriate error message given.

4.2.1 Grid Reference Technique of Referencing a Point

A six figure Grid Reference (GR) technique of referencing a point on the map, which is commonly used in Military usage, has been used for interaction between the user and the system. GR is a 6-digit integer giving location of a point on the map. A GR 457695 represents a point on the map with coordinates (45.7, 69.5) using the Grid coordinate system as depicted on the map. Whenever a user has to specify a point or the system gives a reply to a query in which a point is involved, 6-figure GR is used.

4.3 Organisation of Package

The package is divided in 3 functional modules. These modules are independent of each other and perform specific tasks. Module CREATETREE constructs Strip Tree for a feature (or set of features) for specified w_r.

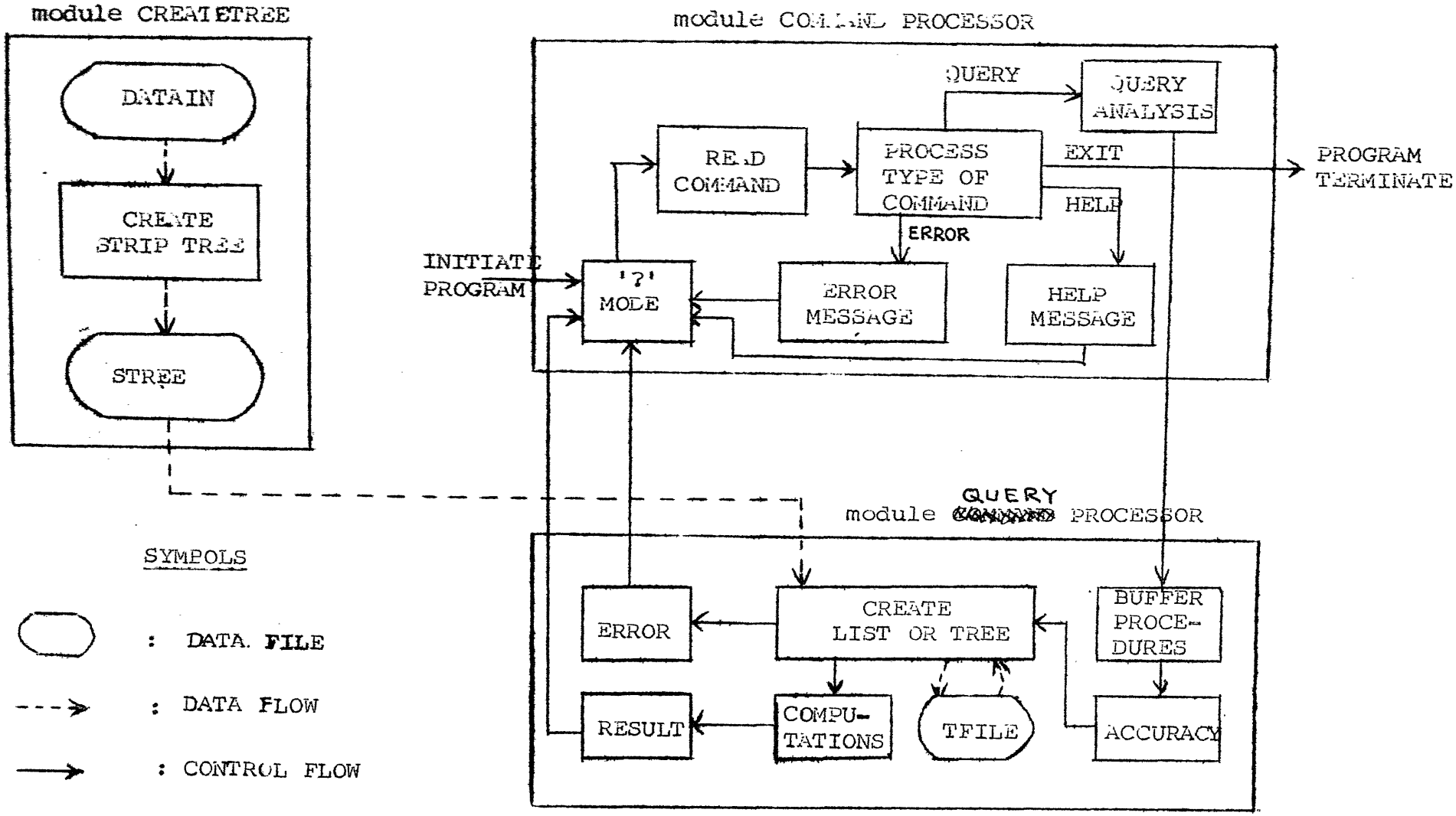


Fig 4.1: Functional Organisation of Graphical Query Processor

Module COMMANDPROCESSOR consists of Command Language Processor (CLP). It analyses the command and passes control to appropriate part of QUERYPROCESSOR module. Finally, QUERYPROCESSOR module performs all the necessary computations according to the nature of query and gives results. The modular layout of system provided flexibility to modify any module without affecting others. Details of implementation of each is described in succeeding paragraphs. Fig. 4.1 shows the functional organisation of the package.

4.4 Module CREATETREE

This module takes digitised data of a feature as the input and creates corresponding Strip Tree, in the form of a binary tree. The leaf node strips of binary tree have width $\leq w_r$, which can be specified by the user. This module is in the form of an independent program and makes use of the fact that creation of Strip Trees is a one time process. In other words, Strip Trees need to be constructed only once before making use of other two modules. Thereafter, this module is not required until a Strip Tree corresponding to some additional feature is to appended, or the resolution of existing ones is required to be modified.

Basis of chosing algorithms and data structure is guided by the fact that it is only a 'one time use' module. Accuracy, rather than efficiency in time, is of more importance, since all further query processing is to be carried out on Tree structure.

The module uses 'Optimum Strip Method' for making strips at each node and 'Mid Point Binary Tree' for Strip Tree representation.

4.4.1 Data Structure

The computations in this module are performed on the set of points describing the feature. Strips formed at each node have all the necessary information which is required for subsequent computations.

A buffer PT, which is a 2-dimensional array of size [1000x2], is used to store x and y coordinates of points describing the curve. These points are obtained from file DATAIN, for one feature at a time, and stored in this buffer. PT[I,X] and PT[I,Y] give x and y coordinates respectively of the I^{th} point on the feature. This indexing of points is useful while dividing the set (or subset) of points in two halves as required for Mid Point Binary Tree Method.

Internal representation of Strip Tree is done by using a linked list. Each node of the TREE contains

4.4.1.1 Freelist

The program is capable of processing data for any number of features. In such a case, unless the nodes used for each feature are not released for the subsequent use, they will keep getting accumulated. The size of storage space required will be

$$O\left(\sum_{i=1}^{n} N_i\right)$$

where N_i is the number of nodes created for i^{th} feature at given resolution, and n is number of features. This size will increase with increase in number of features.

Maintaining a FREELIST of nodes, overcomes this problem. After the Strip Tree for a feature has been built, it is stored onto file STREE and all the nodes used in the TREE are released to join a list, called FREELIST. Whenever a node is required for adding a node to Strip Tree (except in case of ROOT node of a tree, it will always be a pair of nodes corresponding to left and right sons), first a check is made in FREELIST. In case the list is not empty, node is taken from FREELIST, otherwise it is asked from system, using new (P). The order of space required for nodes is O (max N_i). This is not dependent upon number of features.

4.4.2 Building a Strip Tree

Optimum Strip Method as discussed in Chapter II is used for building a strip for a given set of points. The set of points is specified by the indices of first and last point in array PT. The details of some of the important procedures/functions in this module are given below.

(a) Procedure DIRECTION (INC:real;FIRST,LAST:integer);

This procedure finds the direction of optimum strip for points which have indices from FIRST to LAST in Buffer array, PT. The procedure computes width of strip for these points at different angles between MINANGLE and MAXANGLE (these are global variables), at an incremental step given by INC. This procedure is called twice for every set of points. First time it is called with values of MINANGLE and MAXANGLE as 0.0° and 180.0° respectively, and value of INC as 10.0°. If θ is the direction in which strip has minimum width in first iteration then in second iteration, value of MINANGLE and MAXANGLE is set as $\theta \pm 15.0°$ and INC as 1.0°, and process repeated. This gives an optimum strip within an error of $\pm$ 1.0°.

(b) Function FWIDTH (THETA:real;FIRST,LAST:integer): real;

This function is called by procedure DIRECTION to compute the width of strip for a given set of points (indices FIRST to LAST) and an angle THETA. It computes distance of all the points from line -

$$y = mx$$

where m = tan (THETA).

(c) Procedure MAKESTRIP (FIRST,LAST:integer);

This procedure calls procedure DIRECTION with two sets of values of MINANGLE, MAXANGLE and INC as discussed earlier and builds an optimum strip in direction computed in second iteration of DIRECTION. It also computes four corner points of the strip.

(d) Procedure CREATREE (FATHER:PTR;FIRST,LAST:integer);

It is a recursive procedure to build the Strip Tree. Each time, it makes a strip for points with indices FIRST to LAST by making call on MAKESTRIP. If the width of this strip is w_r then it makes a recursive call on itself to create left and rightsons as -

CREATREE (LSON, FIRST, I)

and CREATREE (RSON, I, LAST)

where

I = (FIRST + LAST) DIV 2.

In this manner it creates a depth-first binary Strip Tree.

(e) Procedure PRINTNODE (P:PTR);

This is also a recursive procedure which outputs the Strip Tree nodes onto file STREE. The data in file STREE is stored with first line as feature identification, followed by nodes. Identification line has the format:

Feature class	Feature Ser. No.	NODES	COUNT	MINWR	MAXWR

Where

Feature Class - Class or feature e.g. RIVER.

Feature Ser. No. - Ser. number of feature in its class e.g. RIVER 5.

NODES - Total number of nodes, including Root node and leaf nodes, in the Tree.

COUNT - Number of digitised points of the curve.

MINWR - w_r, for which the tree is built.

MAXWR - Width of Root node.

Subsequent lines (number of such lines is equal to NODES) contain information about each node. Their format is:

SER. NO.	WIDTH	LENGTH	LSON SER	CORNERS

(f) Other Procedures

Certain other procedures are as under:

- READNAME: identifies the name of feature from data file DATAIN.
- INDATA: reads digitised data from DATAIN and stores in buffer array PT.
- GETNEW: Gets a node either from FREELIST, if available, or from system.
- FREELIST: maintains a list of free nodes.

4.5 Module COMMANDPROCESSOR

As discussed earlier, the package has a user friendly command language (CL). The purpose of the language is to make the query commands as simple as possible, and at the same time powerful enough, to avoid any ambiguities.

The package is designed, keeping in view the fact, that users who are going to make use of it, may not

have knowledge of any programming language. The user should not be burdened with any language specifications or system depended features. The entire package, the Strip Tree Representation of Curves, the depth of resolution etc. should be transparent to the user.

4.5.1 Features of CL and CLP

(i) Commands are simple, with a syntax quite similar to English. For instance, for a query:
'find DISTANCE from point 234569 to a feature ROADNH No. 5;'
corresponding command is
DISTANCE/234569/ROADNH.5

(ii) Abbreviations of feature class and query name are permitted. A feature class can be abbreviated to its first 6 characters, whereas in case of query name it can be abbreviated to its first 4 character. For example, CARTTR for CARTTRACK, DIST for DISTANCE can be used.

(iii) Referencing of points is done using 6 digit Grid Reference (GR), commonly used in Military applications.

(iv) It provides flexibility in the ordering of arguments to query. For example

DIST/246892/RIVER.5

and

DIST/RIVER.5/246892

will be interpreted as same query.

(There are certain exceptions to this, which are discussed later).

(v) 'HELP' Command

This command provides necessary help to the user about usage of this package. Whereas HELP provides help of general nature, HELP.extn where extn is the query name, provides help on specific type of query specified by the extension.

(vi) '?' Mode

Once the exeuction of program begins, the program always remains in this mode. A '?' prompt on TTY indicates that program awaits user command. User command can either be a HELP command or a QUERY command. After every command, the program processes it, and after one of the following two conditions, returns to this mode -

(a) Giving HELP message or reply to the query, as the case may be, or

(b) Giving an error message, if there is an error at any stage.

(vii) 'Exit' Command

This command in '?' mode terminates the execution of program.

(viii) Blank Suppression

CLP suppresses all blanks in user commands.

(ix) Upper Case - Lower Case Alphabets

User can use upper or lower case alphabets. However, CLP converts all lower case letters to upper case and all processing is carried out on upper case alphabets.

(x) User commands are accepted as character strings by CLP, and converted to integer or real number wherever necessary. This eliminates any errors caused by conflicts of data-type in the input, which if any is detected by CLP and reported.

4.5.2 Grammer of CL

BNF grammer of command language (CL) is given at Appendix 'D'.

4.5.3 Implementation

As discussed earlier, CL accepts the user command input as a character string, suppressing all the blanks.

A BUFFER of 80 characters is used to store the user command read from TTY, after supressing all the blanks. Further processing is done on data taken from this BUFFER. '/' is used as delimeter between query name and arguments. The nature of query is then obtained from the BUFFER. Following important procedures/functions are used upto this stage:

- procedure READLINE: Reads command from TTY and stores in BUFFER.
- procedure TESTQUERY: Tests for nature of query/HELP command. In case of HELP or HELP.Extn, gives appropriate Help message. In case of a query, gets a serial number corresponding to the query name.
- function FINDQN: Returns query number as an integer corresponding to the serial number of the query in the ordered list of query names, as given at Appendix 'E'.

4.5.3.1 Code Generation

Once CLP has obtained the information regarding the nature of query (not a HELP command), it has to

obtain its arguments and process them for their validity etc. The arguments which follow the query name can be of following types:

(a) GR.of A Point:

It is a 6-digit integer number corresponding to location of a point.

(b) Feature Name

It has name of feature-class, followed by the Serial number of feature under that class, e.g. in case of RIVER No. 7, RIVER is the name of feature class to which it belongs and, 5 its serial number under the class. This is specified as RIVER.5 in the command. When any or all of the features under a feature-class are to be referred, ANY or ALL is given in place of serial number (e.g. RIVER.ANY or CARTTRACK.ALL). The feature can further be subdivided into 2 groups:

(i) Open Features - River, Road etc.

(ii) Close Features - Contours, Boundaries of areas etc.

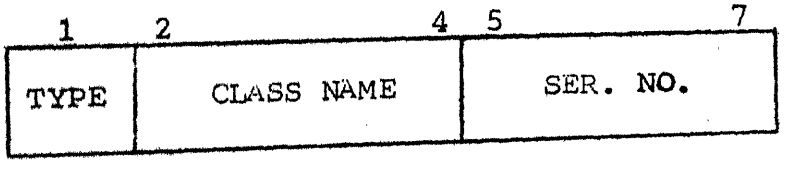
This distinction is made after an op-code is generated corresponding to the query (explained later).

(c) Special Arguments

Certain special arguments which pertain to specific query or argument can be given. They are:

- INSIDE: To specify portion of a feature lying inside a closed feature.
- ENTIRE: In LENGTH query, when the length of entire feature is required.
- MAG/GRID/TRUE: To specify the type of Bearing to be computed.

A 7-digit integer op-code is then generated for each argument. The code has following information:



Digit 1 - First digit of op-code gives type of argument, as given below:

0 : a 6-digit integer GR

1 : an open feature

2 : a closed feature

3 : special argument

Digit 2 to 4 - Digits 2, 3 and 4 give serial number of feature-class in the ordered list of feature-class, or the serial number

of special argument from the ordered list of special arguments, as given at Appendix 'E'. In case of Type GR, these digits combined with digit 5-7 give 6-digit GR.

Digits 5 to 7 - In case of a Type Open or closed feature, digits 5-7 contain the Ser. No. of feature in its class. In case of ANY or ALL, value of these digits in 000 and 999 respectively. For Type Special arguments, value is 000, and in case of Type G GR, these digits, combined with digits 2-4 give 6-digit GR of point.

Examples:

	Argument		Op-Code
(i)	Point 245659	-	0245659
(ii)	RIVER.5	-	1004005
(iii)	RIVER.ANY	-	1004000
(iv)	COUNTOUR.ALL	-	2001999
(v)	INSIDE	-	3002000
(vi)	MAG	-	3003000

4.5.3.2 Analysis of Query

This part of module COMMAND-PROCESSOR, carries out an analysis of arguments along with the nature of query asked for. It performs following taks:

(i) Tests for correct number of arguments as per the type of query.

(ii) Tests whether arguments are compatible with type of query. For instance, in case of query 'HEIGHT', argument should only be a point GR.

(iii) In case of argument as feature name, it tests whether feature should be open or closed feature.

(iv) Tests whether ordering of arguments, where flexibility of ordering is not permitted, is correct.

After testing various conditions, it re-orders the arguments where the flexibility to the order is allowed, and the same has been changed. It then makes call on one of the 'Buffer' procedures with appropriate arguments and passes control to module QUERY-PROCESSOR.

4.6 Module QUERY-PROCESSOR

This is the part of package which performs all the necessary computations to answer the query. It is initiated by one of the 'Buffer' procedures called by COMMAND-PROCESSOR module. Each of these buffer procedure corresponds to a specific type of query as specified by set of commands in Appendix 'B'. The module makes use of data

in Strip Tree form in file STREE, and creates an internal representation of Strip Tree or list of leaf nodes upto those nodes of Strip Tree in STREE which are just $\lesssim w_r$, specified by user for computations.

4.6.1 Data Structure

As in the case module CREATTREE, a linked list using pointers, has been used for internal representation of the TREE or LIST of the nodes required for the computations. Only one array -

CHECKTABLE: packed array [1...500] of boolean;

is used, whose purpose is explained later.

4.6.2 Implementation

As discussed earlier, depending upon the nature of query, either a LIST of those nodes in the tree which are just $\lesssim w_r$, or an internal representation of the Strip Tree upto the resolution w_r is required to be built. The purpose and algorithm to build a LIST or TREE is explained.

4.6.2.1 LIST of Nodes

For the queries, DIST, LENG, CROS (where point of intersection is also required), AREA, HEIG, INTE and CONTAIN, a LIST of those nodes in the Strip Tree, which

contains strips of width $\lesssim w_r$, is required. This is maintained as a Linked List.

Algorithm used, performs a sequential search of nodes starting from the ROOT node. Whenever a strip which has the width w_r is encountered, it is made to join the LIST. CHECKTABLE is used to ensure that, those strips which are SONS of a node joining the LIST, and have width $\lesssim w_r$, are not included, when encountered subsequently. This is achieved by making entry corresponding to such SONS as 'false' in the CHECKTABLE.

4.6.2.2 Creation of TREE upto Resolution 'w_r'

Certain queries require the tree structure of the strips for efficient computations. These are CONTAINMENT and CROSSING. In order to have an internal representation of Strip Tree using linked list, an auxiliary file TFILE is requird, where entire Strip Tree corresponding to the feature on which computations are to perfomed, is copied from file STREE. This is essential since the records are required not in sequential order of their existence in the file, but as per the Tree Structure, where processing is done in order of LSON and RSON, recursively. The records are accessed more than once and appropriate ones selected in each sequence of processing.

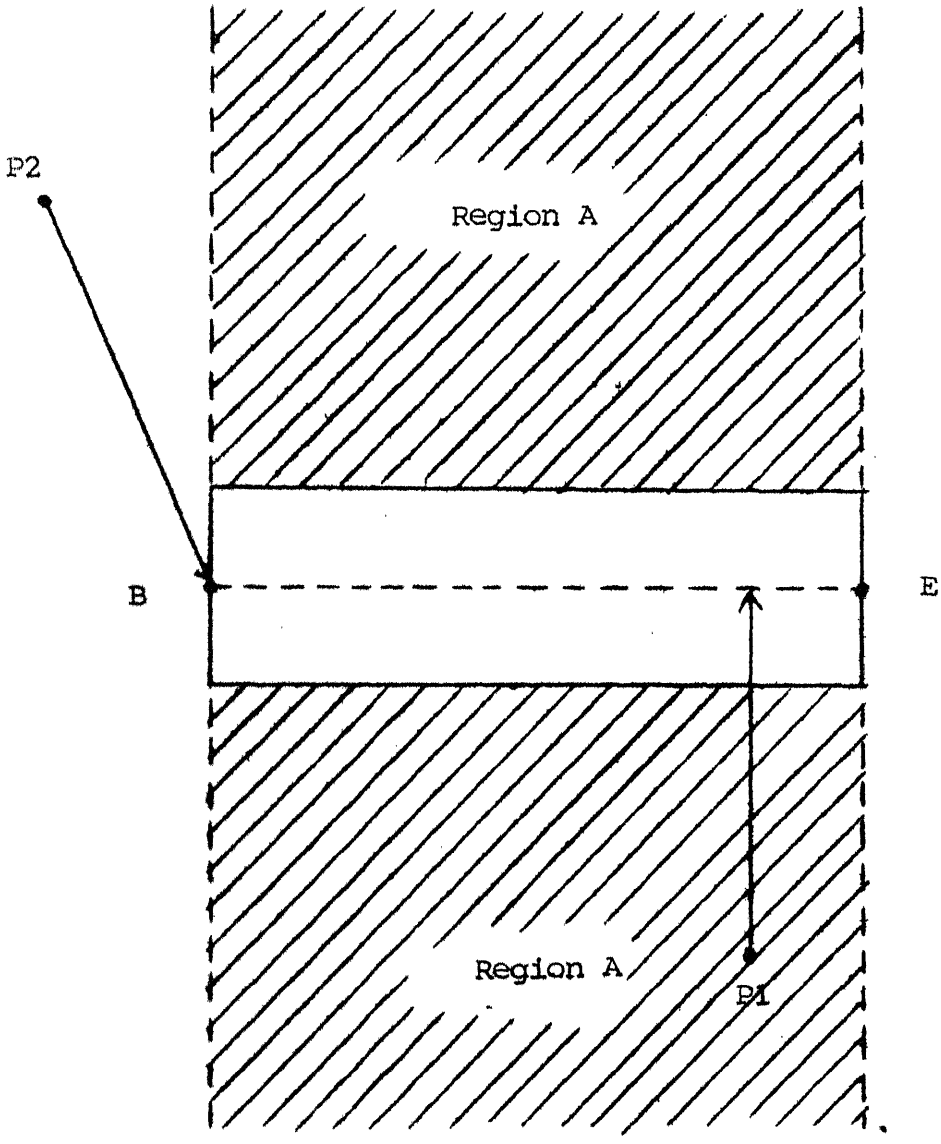


Fig 4.2: Computation of Distance of a Point to a Strip

4.6.3 Some Important Algorithms

The basic operations on the Strip Trees were discussed in Chapter III. Essentially all the computation to answer various queries make use of one or more of these basic operation. Certain queries which do not involve any feature, but only the point or points, do not make use of Strip Trees. These are:

(a) Distance of a point to another point

(b) Mag, True or Grid bearing of a point to another point.

The algorithms for various queries are briefly discussed.

4.6.3.1 Distance

For computing distance of a point to a feature, at resolution w_r, LIST of nodes is made as discussed earlier. Distance from point to each of the strips in the LIST is computed, and minimum of these distances is reported as the result.

To compute the distance of point to a strip, it is computed as perpendicular distance to the line B-E, if the point is in region 'A' of the strip (point P1) or minimum of the distance from point B and E, in case of point lying outside the region 'A' (point P2). (Refer to Fig. 4.2).

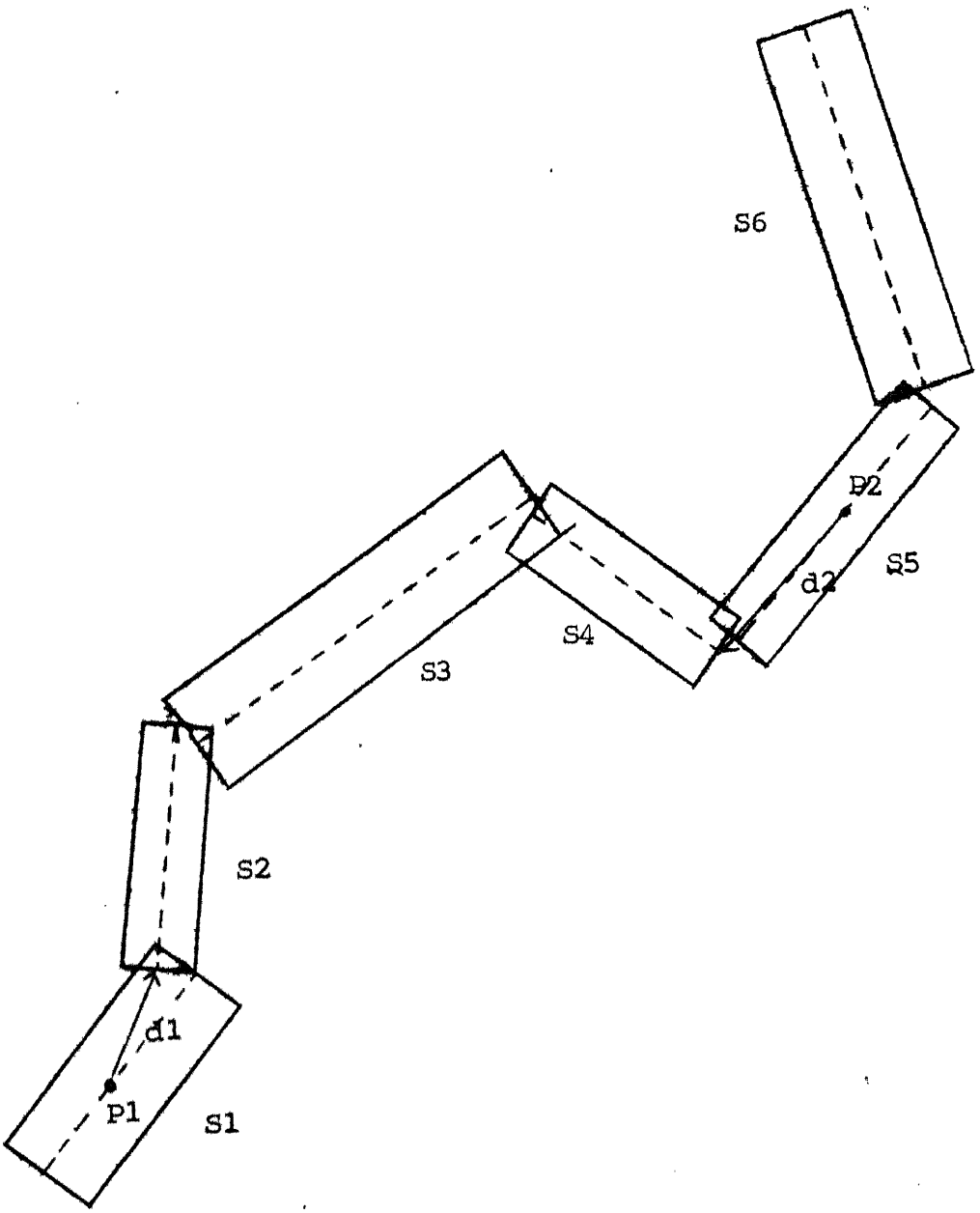


Fig 4.3: Length of a Portion of a Feature between Points P1 and P2

4.6.3.2 Length

As discussed earlier in Section 3.4.2, the error in length calculation can be quite unpredictable if strips are used. To avoid this problem, the actual length of the curve, for the portion enclosed in its corresponding strip, is also stored in one of the fields of the strip.

In order to calculate length, it is summed up from the LIST of strips. Only in case, where the length of portion of curve is required, a modification is required. The two points which describe the portion of feature will specify the two nearest strips from the LIST. The length from these two strips to the next strips on both sides is computed as the shortest distance between the point and the next strip (Fig. 4.3), and adding the length of remaining strip as explained earlier. The length of curve as shown in Fig. 4.3 will be -

length = d_1 + length information of strips S_2 + length information of strip S_3 + ... + d_2

where d_1 is minimum distance from point P_1 to strip S_2

d_2 is minimum distance from point P_2 to strip S_4.

4.6.3.3 Intersection (Crossing)

It requires tree structure to compute whether two features intersect each others or not. Algorithm discussed in Section 3.4.3 is used.

In case points of Intersection are also required, it is obtained by finding point of intersection between each strip of LIST1 (corresponding to first feature) and every strip of LIST2 (corresponding to second feature).

4.6.3.4 Test for Containment

Algorithm of Section 3.4.4 is used. Semi-infinite line S_0 is formed by chosing other point as (100.0, 100.0). Number of intersections is calculated as explained in Section 4.6.3.3, and result obtained, depending upon whether this number is odd or even.

4.6.3.5 Bearing

'Grid Bearing' from Point P1 to Point P2 is defined as, the inclination of vector $\overline{(P1 - P2)}$ with vertical grid lines on the map. Thus

Bearing (P2, P1) = Bearing (P1, P2) + 180°

variations of Magnetic Bearing and True Bearing is specified on each map. Hence these can be computed from Grid Bearing.

4.6.3.6 Height and Intervisibility

Height of a point can be computed by testing its containment within the set of contours on the map. The height will be given by the contour with maximum height, which contains this point.

The method involves testing containment of point, and once containment is found, to extract the information about height of the contour. Presently, the implementation of contours in creation of map [2] is not organised to provide the height information and therefore the computations for height can not be performed. Once such an information is associated with contours, it can be implemented. However processing on HEIGHT command, and testing of argument has been implemented, and control is passed to appropriate Buffer procedure, which simply gives a message.

Similarly computation of intervisibility involves computation of heights of the two points. In addition, height of highest point on the line joining two points is required. This has also been implemented only upto Buffer procedure level.

4.6.3.7 Area of a Closed Feature

Like length, area of curve is also computed at the time of creating Strip Trees in module CREATTREE. This is stored in the identification line of each feature (In

case of open features the values is 0.0). To answer this query in 'Query Process' module, this value is obtained from the identification line of the feature.

Fig 5.2: RIVER1 Shown at 3 Differ[illegible]olutions
(a) 0% (b) 50% (c) 70%

CHAPTER V

CASE STUDY AND CONCLUSION

5.1 Case Study

The package has been tested using the base map [2], for all the queries except HEIGHT and INTERVISIBILITY, for the reasons explained in Chapter IV. Fig. 5.1 shows the map used for this purpose. Some examples about the nature of queries and corresponding answers, are shown in Appendix 'F'. Fig. 5.2 shows the Strip Tree Structure for RIVER-1 at different resolutions.

5.2 Summary

The present work is a general purpose package which can be used for geometrical query processing where curves are represented by set of points, instead of mathematical relations. The approximation using Strip Tree representation, makes the computation process efficient where a large set of points can be approximated to a small member. Some of the queries e.g. containment, intersection, where the answer is negative (i.e. point is not contained in a curve or the curves do not interact) is resolved by first root node itself.

5.3 Suggestions

(i) Presently creation of base map [2], answering of non-graphical queries [4], and this work has been done independently as separate package. Integrating it into one package will increase its usefulness.

(ii) As discussed earlier, the contours on the base map generated, do not have the height information associated with them. If the necessary modification is made to include height information, queries of HEIGHT and INTERVISIBILITY can also be resolved.

(iii) At present, answers to the queries are given in alpha-numeric form only, and they can not be displayed graphically. This can be made possible with raster graphics devices, where selective display and eraser is possible. Whenever the answer to a query is some feature, it can be indicated on the displayed map using flashing arrows.

(iv) The concept of Strip Trees can be extended to three dimensions to approximate heights, using Strip Boxes. This may be useful in 3-dimensional Sand Model displays.

(v) The package has used sequential files, where a sequential search for data pertaining to the required feature is not the most efficient solution. Suitable memory management using lower level language, and organising random access data files, the package can be made more efficient.

REFERENCES

[1] Monmonier, Mark S., "Computer-Assisted Cartography Principles and Prospects", Prentice Hall, 1982.

[2] Subramanian, Capt A.V., "Cartographic Considerations in Computer Aided Sand Model Simulation", M.Tech. Thesis, Submitted to Computer Science Department, IIT Kanpur, Dec 1982.

[3] Chopra, Maj L.K., "Computer-Aided Sand Model Picture Creation Using Interactive Mode", M.Tech Thesis, Submitted to Computer Science Department, IIT Kanpur, Aug 1981.

[4] Surendra Nath, Capt K., "Query Processing for Military Maps and Sand Models (Non-Graphical Queries)", M.Tech Thesis, Submitted to Computer Science Department, IIT Kanpur, Aug 1983.

[5] Ballard, Dana H., "Strip Trees: A Hierarchical Representation of Curves", CACM, Vol 24, Number 5, May 1981, pp 310-321.

[6] Kreyszig, Erwin, "Advanced Engineering Mathematics", 2nd ed., Wiley Eastern Pvt. Ltd., New Delhi, 1969.

[7] Minsky, M.L. and Papert, S., "Perceptrons: An Introduction to Computational Geometry", MIT Press, Cambridge MA, 1969.

[8] William M. Newman, Rober F. Sproull, "Principles of Interactive Computer Graphics", Second Edition Mc-Graw Hill, Kagakusha, 1980.

APPENDIX 'A'

TYPE OF GRAPHICAL QUERIES

1. Distance from a point to another point or a feature
2. Length of a given feature
3. To test whether two features intersect each other
4. To test whether a point is contained inside a closed feature
5. Area of a closed feature
6. Height of a given point from contour information on the map
7. To test whether two points are intervisible to each others.
8. To find Magnetic, Grid or True bearing between two points.

APPENDIX 'B'

LIST OF GRAPHICAL QUERIES AND CORRESPONDING USER COMMANDS

S. No.	Type of Query	Specification of Query	User Command
1.	DISTANCE	(i) Find distance between point ⟨GR⟩ and point ⟨GR⟩	DIST/GR/GR
		(ii) Find shortest distance between a feature ⟨featurename⟩ and point ⟨GR⟩	DIST/featurename/GR
		(iii) Find nearest ⟨featuretype⟩ from point ⟨GR⟩	DIST/featuretype ANY/GR
		(iv) Find shortest distance from point ⟨GR⟩ to all the features of ⟨feature type⟩	DIST/featuretype ALL/GR
2.	CONTAINMENT	(i) Is point ⟨GR⟩ enclosed in closed feature ⟨closed featurename⟩	CONTAIN/GR/closed feature name

		(ii) Is point ⟨GR⟩ enclosed in any of the ⟨closed featuretype⟩	CONTAIN/GR/closed featuretype ANY
3.	CROSSINGS	(i) Does feature ⟨feature name⟩ cross feature ⟨feature name⟩	CROSS/featurename/featurename
		(ii) Does feature ⟨featurename⟩ cross any of features ⟨feature type⟩	CROSS/featurename/featuretype ANY
		(iii) Does any of the features ⟨feature type⟩ cross any of the features ⟨feature type⟩	CROSS/featuretype ANY/featuretype ANY
4.	LENGTH	(i) Find length of feature ⟨feature name⟩ between two points ⟨GR⟩ and ⟨GR⟩	LENGTH/featurename/GR/GR
		(ii) Find length of entire feature ⟨feature name⟩ inside the given map	LENGTH/ENTIRE/featurename
		(iii) Find length of feature ⟨feature name⟩ for the portion lying inside a closed feature ⟨closed feature name⟩	LENGTH/INSIDE/featurename/closed featurename
		(iv) Find length of feature ⟨feature⟩ for the portion lying inside any of the closed feature ⟨closed feature type⟩	LENGTH/INSIDE/featurename/closed featuretype ANY

		(v) Find leggth of any feature ⟨feature type⟩ for the portion inside closed feature ⟨closed feature name⟩	LENGTH/INSIDE/feature-type ANY/closed feature-name
		(vi) Find length of any feature ⟨feature type⟩ for the portion inside any of the closed feature ⟨closed feature type⟩	LENGTH/feature type ANY/ closed featuretype ANY
5.	AREA	(i) Find area of closed feature ⟨closed feature name⟩	AREA/closed featurename
6.	HEIGHT	(i) Find height of point ⟨GR⟩	HEIGHT/GR
7.	INTERVISIBILITY	(i) Are two points ⟨GR⟩ and ⟨GR⟩ inter-visible	INTERVIS/GR/GR
8.	BEARING	(i) Find Magnetic Bearing from Point ⟨GR⟩ to point ⟨GR⟩	BEARING/MAG/GR/GR
		(ii) Find Grid Bearing from Point ⟨GR⟩ to point ⟨GR⟩	BEARING/GRID/GR/GR
		(iii) Find True Bearing from Point ⟨GR⟩ to point ⟨GR⟩	BEARING/TRUE/GR/GR

Notes: 1. GR is 6-figure grid reference of point on the map, as used in all Military applications. GR 694585 means point (69.4, 58.5) on the map.

2. Feature-type means only the class of feature e.g. RIVER, ROADNH etc.

3. featurename means class of feature along with serial number of feature under reference, i.e. RIVER.5 means River no. 5, CARTTRACK.2 means cart track no. 2.

4. closed feature name means only a feature which is defined by a closed curve, e.g. FOREST.2, CONTOUR.3 etc.

5. Wherever there are two or more arguments in a command, they can be given in any order except for LENGTH/INSIDE and BEARING where ordering is necessary.

6. Value inside ( ) is actual value of type specified.

APPENDIX 'C'

TYPES OF ERRORS DETECTED

1. Illegal query type.
2. Illegal argument.
3. No argument to query.
4. No HELP available on specified query.
5. Too few arguments for this query.
6. GR contains non-integer characters.
7. GR is not a 6-figure GR.
8. Illegal feature name.
9. Feature name incomplete.
10. Given feature name does not exist on map under reference.
11. Both arguments are same point - Bearing calculation not possible.
12. Argument for this query should be a closed feature.
13. Both arguments for this query should be points.
14. Arguments for this query should be - A point and a feature name.
15. Arguments for this query should be - A point and a closed feature.

16. Only argument for this query should be - point.

17. Both arguments for this query should be - feature names.

18. Argument for this query should be - closed feature.

19. For LENGTH - if two arguments are points third should be feature name.

20. Ambiguous INSIDE command.

21. Ambiguous ENTIRE command.

22. Illegal arguments for LENGTH.

23. Illegal arguments for BEARING.

24. Computations not possible.

APPENDIX 'D'

BNF GRAMMER FOR COMMAND LANGUAGE

COMMAND ::= ⟨QUERYNAME⟩ ⟨ARG⟩ ⟨ARG⟩ ⟨ARG⟩

QUERYNAME ::= ⟨HELP⟩ | ⟨HELP⟩ ⟨.⟩ ⟨QUERYTYPE⟩ |
⟨QUERYTYPE⟩

QUERYTYPE ::= ⟨DIST⟩ | ⟨LENGTH⟩ | ⟨CROSS⟩ | ⟨AREA⟩
⟨CONTAIN⟩ | ⟨BEARING⟩ | ⟨HEIGHT⟩ |
⟨INTERVISIBILITY⟩ | ⟨DISPLAY⟩

ARG ::= ⟨ ⟩ | ⟨/⟩⟨ARGUMENT⟩

ARGUMENT ::= ⟨POINT⟩ | ⟨FEATURECLASS⟩ ⟨.⟩ ⟨EXTN⟩ |
⟨ENTIRE⟩ | ⟨INSIDE⟩ | ⟨MAG⟩ | ⟨GRID⟩ |
⟨TRUE⟩

POINT ::= ⟨DIGIT⟩ ⟨DIGIT⟩ ⟨DIGIT⟩ ⟨DIGIT⟩ ⟨DIGIT⟩
⟨DIGIT⟩

DIGIT ::= ⟨0⟩ | ⟨1⟩ | ⟨2⟩ | ⟨3⟩ | ⟨4⟩ | ⟨5⟩ | ⟨6⟩ | 7
⟨8⟩ | ⟨9⟩

FEATURECLASS ::= ⟨OPEN⟩ | ⟨CLOSED⟩

OPEN ::= ⟨ROADNH⟩ | ⟨ROADSH⟩ | ⟨CARTTRACK⟩ | ⟨RIVER⟩
⟨RAILLINE⟩ | ⟨INTBOUNDARY⟩ | ⟨CANAL⟩

CLOSED ::= ⟨CONTOUR⟩ | ⟨ENOCAREA⟩ | ⟨OWNOCAREA⟩ |
⟨FOREST⟩ | ⟨MARSH⟩

EXTN ::= ⟨DIGIT⟩ ⟨DIG⟩ ⟨DIG⟩

DIG ::= ⟨ ⟩ | ⟨DIGIT⟩

APPENDIX 'E'

ORDERED LIST OF QUERIES, FEATURE CLASSES AND SPECIAL ARGUMENTS

(a) Queries

Ser. No.	Query Name
1.	DISTANCE
2.	LENGTH
3.	CROSSING
4.	AREA
5.	CONTAINMENT
6.	INTERVISIBILITY
7.	HEIGHT
8.	BEARING
9.	DISPLAY

(b) Feature-Classes

Ser. No.	Open Features	Closed Features
1.	ROADNH	CONTOUR
2.	ROADSH	ENOC-AREA
3.	CARTTRACK	OWNOC-AREA
4.	RIVER	FOREST
5.	RAILLINE	MARSH
6.	INTBOUNDARY	-
7.	CANAL	-